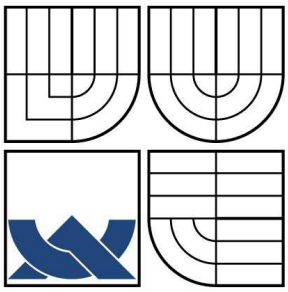
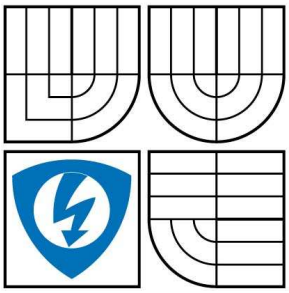


NOCIUPVYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

TEORIE GRAFŮ – IMPLEMENTACE VYBRANÝCH PROBLÉMŮ

GRAPH THEORY - IMPLEMENTATION OF SELECTED PROBLEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. FRANTIŠEK STRÁNÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN KOUTNÝ
SUPERVISOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. František Stráník	***ID:***	83214
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Teorie grafů - implementace vybraných problémů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s algoritmy z oblasti teorie grafů. Na základě poznatků navrhněte a realizujte aplikaci pro vizuální výuku vybraných problémů a algoritmů z teorie grafů. Zejména se zaměřte na problematiku hledání nejkratší cesty v grafu, hledání minimální kostry, počet koster. Pro danou realizaci využijte známých algoritmů. V aplikaci zahrňte jak generátor náhodného grafu, tak možnost tvorby vlastního. Při realizaci se snažte postupovat modulárně tak, aby bylo možné aplikaci jednoduchým způsobem v budoucnu o další problémy snáze rozšiřovat.

DOPORUČENÁ LITERATURA:

[1] DEMEL, Jiří. Grafy a jejich aplikace. Praha : Academica, 2002. 257 s. ISBN 80-200-0990-6.
[2] DEITEL, Harvey, DEITEL, Paul. C++ How to Program (5th Edition) (How to Program). 5th enl. edition. New Jersey : Prentice Hall, 2005. 1536 s. ISBN 0131857576.
[3] ČERNÝ, Jakub. Základní grafové algoritmy [online]. [cit. 2009-02-18]. Dostupný z WWW: <http://kam.mff.cuni.cz/~kuba/ka/ka.pdf>

Termín zadání:	9.2.2009	***Termín odevzdání:***	26.5.2009

Vedoucí práce: Ing. Martin Koutný

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ANOTACE

Tato práce je zaměřena na seznámení se základními problémy z oblasti teorie grafů. Jsou zde popsány základní pojmy i složitější problémy. Jedna část práce je zaměřena na práci s jednotlivými typy grafů. Začíná se s jednosměrně vázaným seznamem, přes obousměrně vázaný seznam až po stromy, které reprezentují nejjednodušší grafové struktury. Další část práce se potom věnuje grafu jako celku a popisuje složitější problémy a jejich řešení. Mezi tyto problémy patří vyhledávání v grafech pomocí metod DFS (Depth First Search) a BFS (Breadth First Search). Dále potom hledání nejkratší cesty za pomoci specifických algoritmů jako jsou: Dijkstrův algoritmus, Floyd-Warshallův algoritmus a Bellman-Fordův algoritmus. Poslední část je věnována problematice vyhledávaní minimálních koster grafu s využití metod Kruskalova haldového algoritmu, Jarníkova (Primova) algoritmu a Borůvkova algoritmu.

Klíčová slova: teorie grafů, vyhledávací strom, BFS, DFS, Dijkstrův algoritmus, Floyd-Warshallův algoritmus, Bellman-Fordův algoritmus, Kruskalův algoritmus, Jarníkův algoritmus, Borůvkův algoritmus

ABSTRACT

This work is intended on identification with basic problems from the graphs theory area. There are the basic conceptions as well more complicated problems described. The one part of this work is specialized in working of individual types of graphs. It starts with single linked list through double linked list after as much as trees which represented the simplest graphs textures. The other part of this work devotes to the whole graph and describes more complicated problems and their resolution from the theory graphs area. Among these problems belongs to searching in graphs help by Depth First Search and Breadth First Search methods. Then searching the shortest way help by the specific algorithms as are: Dijkstra´s algorithm, Floyd-Warshall´s algorithm and Bellman-Ford´s algorithm. The last part is devoted to problems with searching minimal frames of graphs with usage Kruskal´s algorithm, Jarnik´s algorithm and Boruvka´s algorithm methods.

Keywords: graphs theory, search tree, BFS, DFS, Dijkstra´s algorithm, Floyd-Warshall´s algorithm, Bellman-Ford´s algorithm, Kruskal´s algorithm, Jarnik´s algorithm, Boruvka´s algorithm

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Martinovi Koutnému, za metodickou pomoc a rady při vypracovávání bakalářské práce.

V Brně dne …….. ……..
(podpis autora)

Obsah

Seznam obrázků

1 Úvod

Ve výpočetní technice je řešena veliká spousta úloh, která je zaměřena na práci s útvary, které by se daly považovat za grafy. Tyto úlohy potom vyžadují znalost alespoň základních pojmů z teorie grafů. Tato práce má za cíl seznámit čtenáře s těmito základními pojmy a osvětlit jakým způsobem je možné s grafy pracovat. Samozřejmě, že rozsah práce nedovoluje postihnout teorii grafů jako celek. Je kladen důraz na pojmy jako jsou orientovaný a neorientovaný graf, základní typy grafů nebo způsob jakým je možno graf zpracovat za pomocí výpočetní techniky.

Poslední kapitola je věnována práci s abstraktními datovými typy, které samy o sobě reprezentují části nebo celé grafy. Sem patří jednosměrně vázaný seznam, obousměrně vázaný seznam a stromy. Je ukázáno, jakým způsobem je možné prvky do jednotlivých útvarů vkládat, odebírat nebo v nich vyhledávat.

Další část poslední kapitoly je věnována vyhledávání v obecných grafech, což je velmi důležitý pojem nejen pro teorii grafů. Velmi podrobně je zde vysvětleno vyhledávání do hloubky a vyhledávání do šířky, jelikož se jedná o základní způsoby vyhledávání v orientovaných i neorientovaných grafech. Pro přehled jsou samozřejmě uvedeny i jiné metody, které ovšem ze dvou výše zmiňovaných vychází nebo je určitým způsobem rozšiřují.

Předposlední část poslední kapitoly se zabývá problémem vyhledávání nejkratší cesty v grafech. Problém je rozlišen na nezáporně a obecně ohodnocené typy grafů. Dalším typem dělení je rozdělení na grafy orientované a neorientované. Pro jednotlivé skupiny je potom ukázáno i několik konkrétních algoritmů. Jsou to: Dijkstrův algoritmus, Floyd-Warshallův algoritmus a Bellman-Kordův algoritmus.

Závěr poslední kapitoly řeší problematiku vyhledávání koster grafů. Opět zde dochází k dělení problému podle typu grafů na obecné a jednoznačně určené. Jako příklady pro určení minimální cesty jsou použity tyto algoritmy: Kruskalův hladový algoritmus s vyhledáváním pomocí přidávání i odebíraní hran, Jarníkův (Priamův) algoritmus a posledním zástupcem v této oblasti je Borůvkův algoritmus.

2 Úvod do teorie grafů.

Graf se skládá ze dvou základních částí.

- Vrcholy – reprezentují v grafu určitou hodnotu nebo stav, do kterého se dostaneme při dodržení nějakých podmínek, nebo naopak stav, do kterého se můžeme dostat naprosto náhodně.
- Hrany – jsou „cestami“ v grafu. Hrana může spojovat dva vrcholy a nebo jeden vrchol sám se sebou. V takovém případě hovoříme o **smyčce.** Dalším typem hran může být tzv. hrana **orientovaná.** U takovéto hrany rozlišujeme její počáteční vrchol a její koncový vrchol. Naproti tomu hrana **neorientovaná** nemá definován ani počáteční, ani koncový bod.

V závislosti na předchozích informacích tedy můžeme rozdělit grafy na několik typů. Dělení probíhá tedy hlavně podle toho, jaké hrany jsou v grafu použity. Jedná se tedy o:

- *Neorientovaný* graf
- *Orientovaný* graf
- *Smíšený* graf

2.1 Neorientovaný graf

Neorientovaný graf vznikne užitím vrcholů a neorientovaných hran. Zvláštností těchto grafů tedy je, že orientace mezi vrcholy není nepodstatná. Takovýto typ grafů se užívá v situacích kdy nemůžeme nebo nechceme rozlišit mezi počátečním a koncovým vrcholem.

Neorientovaný graf je definován jako trojce G = (V, E, ε). Jde o neprázdnou množinu tvořenou konečným počtem prvků V, které nazýváme *vrcholy* grafu. Vrcholy grafu jsou spojovány *neorientovanými hranami* grafu, jež tvoří konečnou množinu E. Poslední z trojce nazýváme *zobrazením incidence*. Jedná se o vztah, který každé hraně e z množiny E přiřazuje jednoprvkovou nebo dvouprvkovou množinu vrcholů grafu G. Tyto vrcholy jsou označovány jako krajní vrcholy neorientované hrany e. Také můžeme říct, že hrana e *je incidentní* s krajními vrcholy nebo že tato hrana vrcholy *spojuje*. Pokud hrana e spojuje jeden vrchol se sebou samým, tak o této hraně říkáme, že je *smyčkou*.

2.2 Orientovaný graf

Orientovaný graf vznikne užitím vrcholů a orientovaných hran. Tento typ grafu je základní, protože velmi často potřebujeme definovat počáteční a koncový vrchol, jež jsou spojeny hranou. Užití tohoto typu grafů je v situacích, kdy je nutné specifikovat pořadí vrcholů.

Orientovaný graf je definován jako trojice G = (V, E, ε). Jde o neprázdnou množinu tvořenou konečným počtem prvků V, které nazýváme *vrcholy* grafu. Vrcholy grafu jsou spojovány *orientovanými hranami*. Hrany tvoří konečnou množinu E. Třetím z trojice definující orientovaný graf je *vztah incidence* ε. Tento vztah přiřazuje každé hraně e z množiny E dvojici vrcholů (x, y). Vrchol x reprezentuje *počáteční vrchol* a naopak vrchol y reprezentuje *koncový vrchol*. Počáteční vrchol hrany e je označován PV(e). Koncový vrchol označujeme KV(e). O hraně e tedy říkáme, že je *incidentní* s vrcholy x a y, když začíná je její počáteční vrchol x a koncový vrchol y. Hrana e potom tyto vrcholy *spojuje*. Jako u neorientovaného grafu i zde může nastat situace, kdy hrana spojuje vrchol se sebou samým. Vznikne tedy situace PV(e) = KV(e). Takovou hranu e potom označujeme jako *orientovanou smyčku*.

2.3 Smíšený graf

Smíšený graf vznikne kombinací orientovaného a neorientovaného grafu. Takovýto graf se užívá v situacích, kde se nedá ve všech případech určit, který vrchol je počáteční a který koncový.

2.4 Nekonečný a prázdný graf

Jedná se o velmi teoretické případy, kdy má graf nekonečně mnoho vrcholů a samozřejmě i nekonečně mnoho hran. Opačným případem je situace, kdy není v grafu žádný vrchol a žádná hrana.

2.5 Vztah mezi orientovaným a neorientovaným grafem

Orientovaný a neorientovaný graf jsou užívány na rozdílné úlohy a způsob práce s oběma druhy grafů je také odlišný. V některých případech je ale potřeba převést jeden druh grafu na druhý a naopak. Pro vzájemný převod orientovaných a neorientovaných grafů zavádíme dva pojmy:

- *Syntetizace* grafu
- *Orientace* grafu

2.6 Syntetizace grafu

Syntetizací se nazývá převod orientovaného grafu na neorientovaný. Toho docílíme pouhým převedením orientovaných hran na neorientované. Prakticky se jenom zapomene na orientaci hran. Každá hrana tedy nemá počáteční a koncový vrchol, ale pouze dva vrcholy spojuje.

2.7 Orientace grafu

Orientace grafu je opakem k syntetizaci. Jedná se o případ, kdy chceme z neorientovaného grafu vytvořit graf orientovaný. Orientaci lze provést dvěma způsoby:

- Náhodná orientace grafu – jedná se o případ, ve kterém je orientace nastavena náhodně. To znamená, že každé neorientované hraně se buď náhodně nebo podle nějakého pravidla přidělí orientace a tím se určí její počáteční a koncový bod. V praxi jde pouze o přikreslení šipek k hranám (viz. níže).
- Symetrická orientace grafu – u tohoto druhu orientace grafu probíhá přepočet tak, že je ke každé neorientované hraně přidána druhá. Orientace se potom zvolí na jedné hraně jedním směrem a na druhé hraně směrem druhým. U smyček je situace odlišná. Každá smyčka se nezdvojí, ale jen se jí přiřadí orientace. Získáme tak graf, kde je počet hran zvýšen, jelikož každé dva vrcholy jsou spojeny dvěma hranami s opačnou orientací, což však neplatí pro smyčky.

Z výše uvedených pravidel je patrné, že převod orientovaného na neorientovaný je úloha jednodušší. Proto se také předpokládá, že orientovaný graf je grafem základním a neorientovaný graf je odvozený. Pro zavedení orientovaného grafu jako základního je hned několik důvodů:

- Pokud jsou někomu sdělovány informace o grafu, je vždy jeden z vrcholů zmíněn jako první, i když je posléze uvedeno, že na pořadí nezáleží.

- Při práci s orientovanými grafy je potřeba dělat operace, které nezávisí na orientaci a převod orientované hrany na neorientovanou je jednodušší.

2.8 Kreslení grafů

Velkou výhodou grafů, a to jak orientovaných tak neorientovaných, je možnost si závislosti mezi vrcholy a orientaci hran zakreslit graficky pro lepší představu. Vrcholy můžeme nakreslit podle potřeby jako kolečka, kroužky nebo prosté body. Hrany jsou potom znázorněny pomocí čar spojujících body (vrcholy). Pokud se jedná o orientovaný graf, tak se orientace značí pomocí šipek, které jdou od počátečního vrcholu ke koncovému. Z uvedených informací je jasné, že jeden a tentýž graf lze zakreslit několika různými způsoby.

2.9 Speciální grafy

Grafy můžou nabývat různých podob. V některých případech dochází v grafu k určitým zvláštnostem. Potom vznikají i grafy, které nelze popsat jednoduše jako orientované nebo neorientované. Jedná se o tyto druhy grafů:

- *Bipartitní* graf
- *Úplný bipartitní* graf
- *Úplný orientovaný* graf
- *Úplný neorientovaný* graf
- *Diskrétní* graf
- *Kořenový strom*

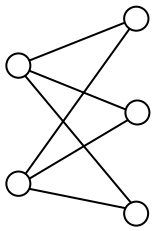
2.9.1 Bipartitní graf

Jedná se o takový graf, kde jsou vrcholy rozděleny do dvou množin S a T. Propojení vrcholů pak probíhá tak, že hrana má jeden krajní bod v množině S a druhý v množině T. Tento typ grafu může být jak orientovaný, tak neorientovaný. Při vytváření orientovaného

bipartitního grafu je potřeba, aby všechny hrany byly orientované stejným směrem (např. aby všechny počáteční body ležely v množině S a koncové body v množině T).

2.9.2 Úplný bipartitní graf

Jde o rozšíření bipartitního grafu. Pro tento typ grafu platí, že vrchol s ležící v množině S a vrchol t ležící v rovině T jsou spojeny právě jednou hranou. Příklad tohoto typu grafu je na obrázku 1.



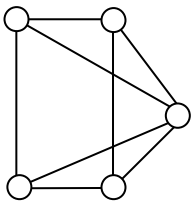
Obr. 1: Úplný bipartitní graf

2.9.3 Úplný orientovaný graf

Tento graf je definován jako G = (V, R), kde R značí množinu všech uspořádaných dvojic vrcholů z množiny V. Jedná se o prostý graf.

2.9.4 Úplný neorientovaný graf

Tento typ grafu je prostý a neorientovaný. Platí pro něj, že každé dva libovolné vrcholy jsou spojeny hranou. V tomto typu grafu se neuvažují smyčky. Příklad úplného neorientovaného grafu je na obrázku 2.



Obr. 2: Úplný neorientovaný graf

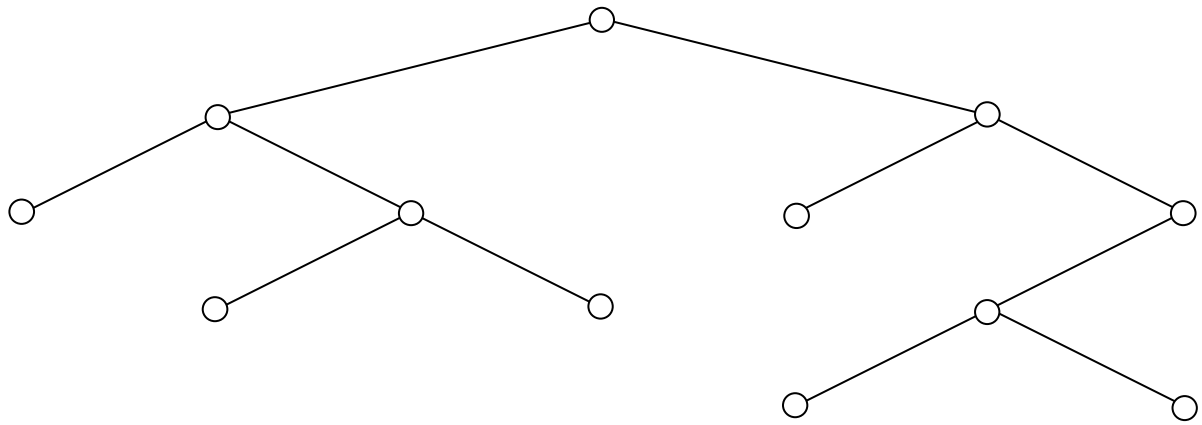
2.9.5 Diskrétní graf

Diskrétní graf nemá žádnou hranu. Výhodou tohoto typu grafu je, že ho můžeme podle potřeby považovat za orientovaný či neorientovaný.

2.9.6 Kořenový strom

Kořenový strom je nejčastějším typem stromu a je používána v nejrůznějších případech a aplikacích.

Jedná se o orientovaný graf, který má jednoznačně určený kořen *r*. Kořen stromu je velmi specifický vrchol, jež nemá žádnou hranu, která by vedla do kořene. Do každého dalšího vrcholu potom vede právě jedna hrana. Příklad jednoduchého kořenového stromu je vidět na obrázku 3.



Obr. 3: Kořenový strom bez vyznačení orientace jednotlivých hran

Kořenové stromy používají zvláštní označení vrcholů. Vede-li z kořenového vrcholu hrana z vrcholu x do vrcholu y, mluvíme o vrcholu jako o *rodiči* a o vrcholu y jako o *potomku.* O vrcholech, které mají společného rodiče se hovoří jako o *bratrech*. Zavádí se i označení vrcholu, který nemá žádného potomka a je tedy posledním vrcholem ve větvi stromu. Takový vrchol je označován jako *list.* Kořen stromu je jediným vrcholem, který nemá rodiče.

- *Hloubka vrcholu* – je definována jako počet hran, které mineme na cestě od kořene do vrcholu y.
- *Vrstva* – jde o množinu vrcholů, které se nacházejí ve stejné hloubce
- *Výška stromu* – je největší počet hran od kořene stromu až k jeho nejvzdálenějšímu listu

2.9.7 Binární kořenový strom

Binární kořenový strom je zvláštní případ kořenového stromu. V tomto stromu má každý rodič maximálně dva potomky. Pro binární kořenový strom potom zavádíme pojmy *levý* a *pravý* potomek.

3 Počítačová analýza grafů

3.1 Způsoby popisů grafů

Počítačová analýza grafů se používá hlavně tam, kde by bylo kreslení grafů velmi zdlouhavé a nepřehledné. Jedná se o situace, kdy je graf velmi rozsáhlý a zpracování pomocí grafické metody by zabralo spoustu času. Pro počítačovou analýzu se používá celá řada postupů, které mezi sebou mají drobné odlišnosti. Každý postup zpracování grafu za pomoci počítače však předpokládá, že vrcholy i hrany jsou očíslovány přirozenými čísly začínajícími od jedničky.

- Nepřímý popis grafu
- Matice sousednosti
- Vektory sousednosti
- Matice incidence
- Matice sousednosti bipartitního grafu
- Seznamy vrcholů a hran
- Seznam vrcholů a seznamy okolí vrcholu

3.1.1 Nepřímý popis grafu

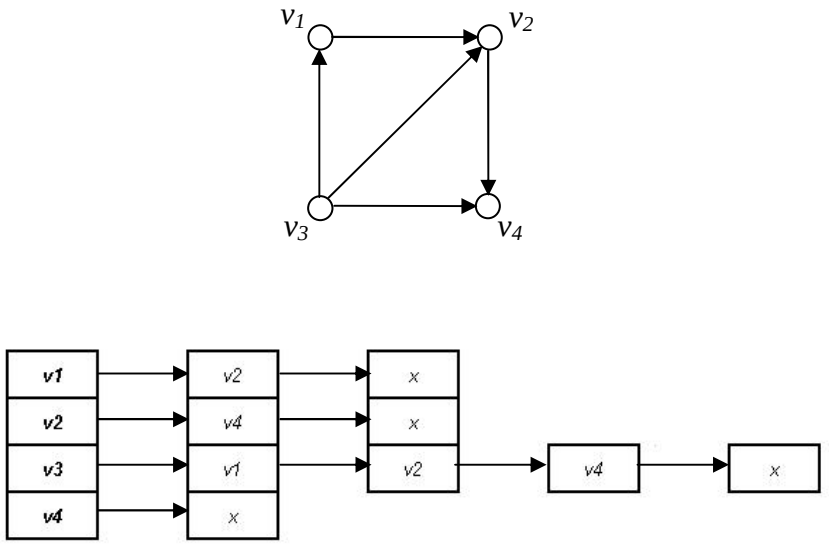
V některých případech není nutné ukládat seznamy vrcholů, hran nebo okolí vrcholů. Graf lze potom zadávat pomocí algoritmů. K procházení grafu potom stačí znát popis kteréhokoliv z vrcholů. Pomocí algoritmu potom procházíme celý graf. Tento postup se používá při zpracování velmi rozsáhlých grafů, které by svým uložením do paměti zabíraly mnoho místa a výpočetního výkonu.

3.1.2 Matice sousednosti

Matematicky elegantní způsob využívající faktu, že matice sousednosti určuje graf až na izomorfizmus. Izomorfizmus je vlastnost grafu, která říká, že dva grafy se liší pouze v označení a nakreslení vrcholů a hran. Tento způsob je velmi neefektivní pro ukládání grafu s malým počtem hran.

3.1.3 Vektory sousednosti

Tento způsob reprezentace grafu je velmi propracovaný co se týče efektivního využití paměti. V paměti jsou totiž uloženy pouze relace obsažené v grafu. Zde je tedy úspora prostředků oproti popisu matice sousednosti. Příklad popisu grafu pomocí vektorů sousednosti je na obrázku 4.



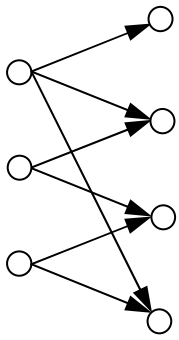
Obr 4: Vektory sousednosti vrcholů orientovaného grafu.

3.1.4 Matice incidence

Pro matici incidence platí stejná pravidla jako pro matici sousednosti s drobnou odlišností. Pokud bychom matici incidence vyplňovali tak, že do každého sloupce zadáme polohy nenulových prvků, dostaneme vlastně matici hran. Jako u matice sousednosti je použitelnost pouze v případech, kdy je graf malý.

3.1.5 Matice sousednosti bipartitního grafu

Jak název napovídá je tato metoda uložení grafu použitelná pouze u bipartitního grafu. Tato matice je založena na faktu, že všechny vrcholy bipartitního grafu lze uspořádat tak, aby matice sousednosti měla velmi specifický tvar. Matice sousednosti se potom tedy nazývá maticí sousednosti bipartitního grafu. Na obrázku 5 je znázorněn orientovaný bipartitní graf a jeho matice sousednosti.



$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Obr 5: Marice sousednosti orientovaného bipartitního grafu

3.1.6 Seznamy vrcholů a hran

Všechny vrcholy jsou zde popsány jednoduchým seznamem (výčtem) prvků. Seznam hran je definován jako seznam uspořádaných trojic. Hrana, počáteční vrchol, koncový vrchol. Pokud se jedná o neorientovaný graf je seznam hran definován jako seznam uspořádaných trojic. Hrana, první vrchol, druhý vrchol. Pokud není podstatný název hrany, tak se seznam může uvádět jako uspořádaná dvojice vrcholů. Tento způsob popisu grafu je velmi univerzální a relativně jednoduchý. Díky těmto vlastnostem je seznam vrcholů a hran velmi hojně používaným způsobem pro uložení grafu do paměti. Vlastnosti této metody také umožňují popisovat a ukládat i ohodnocené grafy. Ohodnocení se prostě napíše k vrcholům nebo hranám.

3.1.7 Seznam vrcholů a seznamy okolí vrcholů

Redukcí předchozího způsobu zadávání grafů dostáváme seznam vrcholů a seznamy okolí. Všechny vrcholy jsou opět popsány jako seznam vrcholů. Hrany jsou zde popsány pomocí uspořádaných dvojic: jméno a název koncového vrcholu. Jméno počátečního vrcholu je známé, protože je stejný pro celou skupinu hran. Tento popis může opět popisovat orientované i neorientované grafy. Pro orientované grafy můžeme vytvořit mnohem redukovanější seznamy. Můžeme totiž vytvořit seznam kladných nebo záporných orientací,

kde každá hrana bude v obou seznamech pouze jednou. Pokud ale vytvoříme pouhý popis okolí vrcholů, bude každá hrana v seznamu dvakrát.

3.2 Zpracování grafu

Mezi popisem grafu a jeho uložením v paměti je veliký rozdíl. Různé platformy či programovací jazyky mohou s grafy pracovat různě. Základní typy zpracování jsou:

- Datové struktury využívající matice
- Datové struktury využívající ukazatele
- Seznamy hran v jednorozměrném poli
- Seznamy následníků v jednorozměrném poli

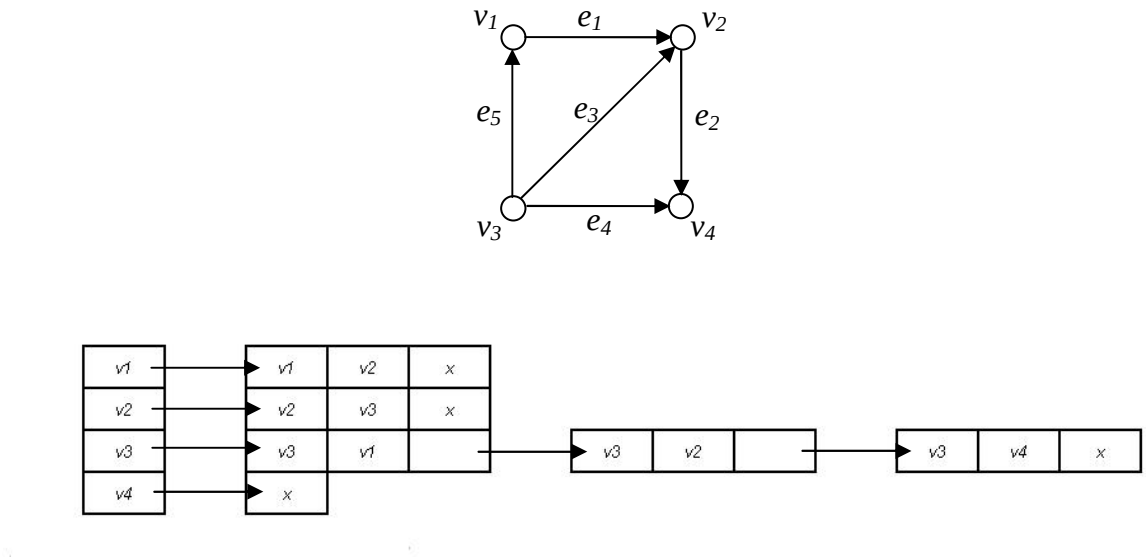
3.2.1 Datové struktury využívající matice

Matice incidence či matice sousednosti nejsou pro grafy s málo hranami příliš vhodné. Jednak jsou zde zbytečně vysoké paměťové nároky a jednak je vyhledávání další hrany, která vychází z téhož vrcholu, pomalé. Datové struktury využívající matice jsou vhodné spíše v situacích, kdy jsou grafy ohodnocené nebo mají hodně hran. Dalším typem úloh, kde se těchto datových struktur využívá je případ, kdy má být výsledkem výpočtu právě matice. Na příslušná místa v matici se zapíše ohodnocení hrany, pokud existuje. Pokud neexistuje, tak se na pozici definující právě tento spoj zapíše nějaká velká nebo naopak nízká konstanta. Tato konstanta musí být zvolena velmi pečlivě s ohledem na to, jaké typy úloh budou nad grafem prováděny. Pro prosté neohodnocené grafy se v tomto případě používá uložení pomocí matice sousednosti, která je převedena matici bitů. Manipulace s jednotlivými bity je mnohem složitější, než s celými čísly, ale to je vykoupeno velmi nízkými paměťovými nároky.

3.2.2 Datové struktury založené na ukazatelích

Modernější programovací jazyky umožňují pracovat pouze s tzv. ukazateli. Ukazatel je zvláštní případ datového typu, kdy informace uložené na adrese proměnné pouze „ukazují“ na adresu, kde se nachází data objektu. Data která popisují vrchol nebo seznam všech vrcholů sdružujeme do záznamů, které jsou často označovány jako recordy. Každý záznam potom

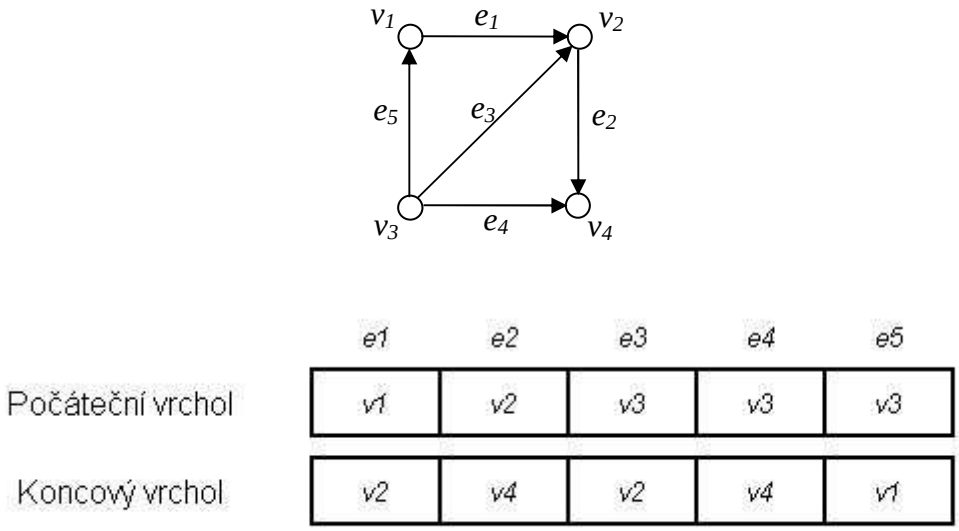
obsahuje ukazatel na záznam následující. Nespornou výhodou této metody jsou vlastnosti ukazatelů. To znamená, že předem nemusíme vědět, jak velký graf budeme ukládat, jelikož přidělenou paměť je možné pomocí ukazatelů dynamicky měnit. Rychlost práce s ukazateli je také nespornou výhodou tohoto způsobu uložení dat v paměti. Pokaždé se pracuje pouze s ukazatelem na data a to je mnohem rychlejší než práce s hodnotami. Velkou nevýhodu metod založených na ukazatelích je množství spotřebované paměti, jelikož pro uložení grafu je zapotřebí paměť na graf, tak i na ukazatele. Občas dochází i ke kombinaci uložení záznamů pomocí ukazatelů a hodnot. Obrázek 6 ukazuje takovou situaci, kdy jsou vrcholy zaznamenány v poli a hrany jsou definovány pomocí ukazatelů, které jsou součástí pole vrcholů. Každá hrana je potom uspořádanou dvojicí počátečního a koncového vrcholu.



Obr. 6: Datový popis grafu založený na ukazatelích

3.2.3 Seznamy hran v jednorozměrném poli

Předpokladem pro použití této metody je očíslování jednotlivých vrcholů i hran pomocí přirozených pořadových čísel. Seznam hran orientovaného grafu lze potom uložit do dvou jednorozměrných polí, kde jako index hrany slouží pořadové číslo. V prvním poli je umístěn seznam počátečních vrcholů a v druhém seznam koncových vrcholů. Pokud by byl tento graf doplněn o ohodnocení hran, tak je možné použít další jednorozměrné pole. Tento způsob uložení je vhodný v programovacích jazycích, kde není tak propracovaná možnost strukturování dat. Grafické znázornění použití jednorozměrného pole pro uložení grafu je znázorněno na obrázku 7.



Obr 7: Uložení grafu do jednorozměrného pole

3.2.4 Seznamy následníků v jednorozměrném poli

Tento způsob uložení prvků v paměti pro svoji práci využívá dvě jednorozměrné pole. V prvním poli je uloženo pořadové číslo prvku, které v druhém poli začíná seznam následníků vrcholu, určeného pořadovým číslem z prvního pole. Následníci jsou v poli následníků uloženi v těsné blízkosti vedle sebe bez jakýchkoliv oddělovačů. Je-li graf uložený pomocí následníků multigrafem, tak jsou rovnoběžné hrany zaznamenány jako opakovaný výskyt následníka. Tento způsob uložení grafu v paměti je velmi úsporný a velmi efektivní co se týče vyhledávání, jelikož velmi snadno můžeme procházet seznam následníků. Přístup k předchozím prvkům je však o něco komplikovanější a to je největší nevýhodou této metody.

4 Popis vytvořených nástrojů

Součástí této práce není pouhé zpracování základních poznatků z oblasti teorie grafů, ale také vytvořit funkční kód, který by pomohl studentům při studiu problematiky grafů. Z toho důvodu byly vytvořeny zdrojové soubory pro několik typů grafů, kde je prakticky ukázáno, jakým způsobem fungují nejrůznější metody pro práci s jednotlivými grafy.

4.1 Vývoj aplikací

Aplikace vzniká ve svém počátku jako prostý text, který má velmi specifické požadavky na formátování. Každý, kdo se učí programovat musí tedy nejdříve pochopit jak program formátovat. S programováním úzce souvisí několik pojmů, které je nutné znát.

- Zdrojový kód
- Kompilování zdrojového kódu
- Linkování
- Debuggování

4.1.1 Zdrojový kód

Zdrojový kód je prostý text, který má určité požadavky na formátování. Proto je možné vytvořit jej v libovolném textovém editoru. Jelikož jsou dále s tímto textem prováděny velmi specifické úpravy, je doporučené použít speciální editor v němž jsou speciální funkce právě pro provádění úprav se zdrojovým kódem. Těmto editorům se říká vývojové prostředí. Následuje velmi krátký a jednoduchý program zapsaný v programovacím jazyce C++.

```
#include <iostream>
void main()
{
   std::cout << „Text“;
   std::cin.get();
}
```

Každý z uvedených řádků má svůj význam a pouhou změnou malých písmen na velké můžeme způsobit, že program nebude fungovat. Základem je zde funkce main, ve které je

zapsán část zdrojového kódu, jež bude proveden po spuštění programu. První příkaz mezi závorkami potom vypíše do konzolového okna nápis „Text“ a druhý vyčká na stisknutí jakékoliv klávesy. Program se potom ukončí.

4.1.2 Kompilování zdrojového kódu

Kompilování zdrojového kódu se nazývá kompilace. To se provádí jedním ze speciálních nástrojů, kterému se říká *kompilátor*. Kompilátor je velmi složitý nástroj s možností velmi důkladného nastavení. Kompilací zdrojového kódu potom vytvoříme objektový soubor.

4.1.3 Linkování

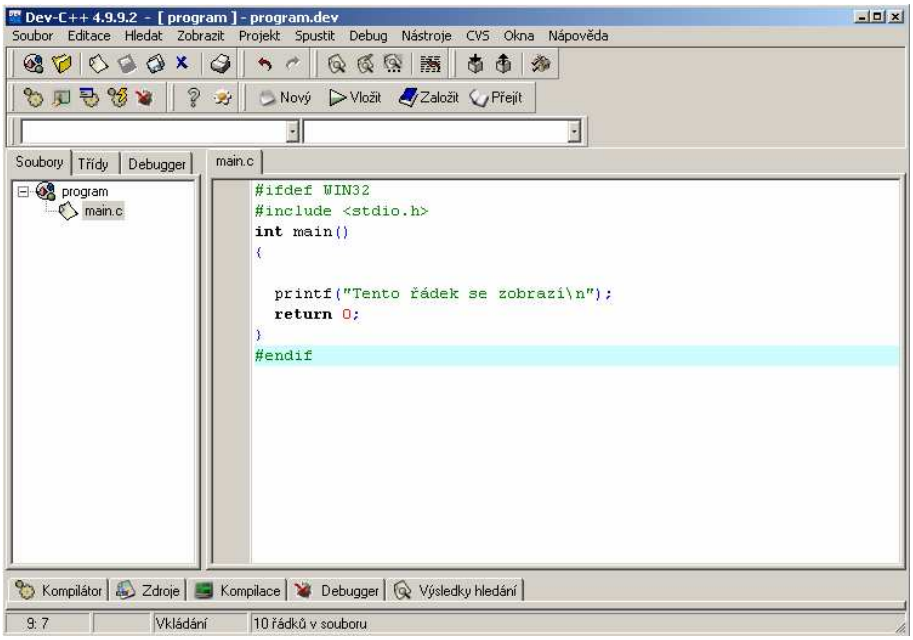
Objektové soubory vytvořené kompilátorem musí být spojeny. K tomuto slouží proces linkování. Kompilátor vytvoří pro každý soubor zdrojového kódu jeden objektový soubor. Linkování potom spojí soubory dohromady (pokud se program sestává z více souborů). Výsledkem je spustitelný soubor.

4.1.4 Debuggování

V případech, kdy zdrojový kód obsahuje jednu nebo více chyb kvůli kterým nelze program sestavit a spustit, je potřeba takové chyby najít a opravit. Pro hledání chyb se používá *debugger*. V případě, že zdrojový kód neobsahuje chyby, je program spuštěn ihned po linkování.

4.1.5 Vývoj aplikací ve windows

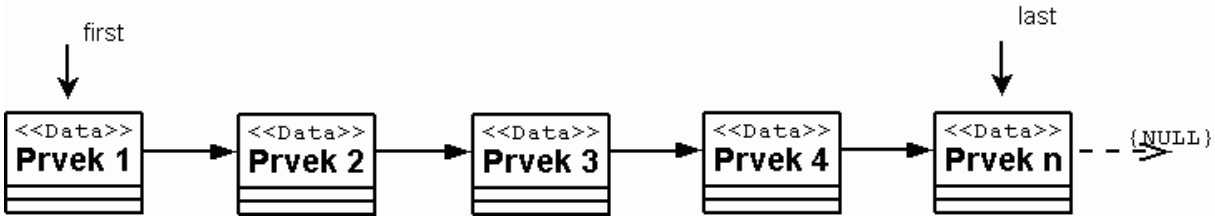
Operační systém windows je určen především pro uživatele osobních počítačů, kteří mají s vývojem aplikací velmi malou nebo žádnou zkušenost. Proto je zde zapotřebí použít již výše zmíněného vývojového prostředí, které v sobě integruje jak editor pro psaní vlastního zdrojového kódu, tak nástroje pro jeho pozdější spuštění nebo detekci chyb. Pro vytvoření některých zdrojových souborů v rámci této práce bylo použito volně dostupného editoru DEV C++. Tento editor je vyvinut programátory operačního systému linux tak, aby bylo možné ho používat na platformě windows. Díky tomu je dostupný zcela zdarma. Okno tohoto editoru je na obrázku 8.



Obr. 8: Okno vývojového prostředí DEV C++

4.2 Jednosměrně vázaný seznam

Jednosměrně vázaný seznam je jedním z nejjednodušších abstraktních datových typů, které bývají v literatuře označovány za větev orientovaného grafu. Každý prvek seznamu nese jak informaci o datové složce, tak i ukazatel na bezprostředně následující prvek. Poslední prvek seznamu potom neukazuje nikam, nebo-li ukazuje na NULL. Příklad jednosměrně vázaného seznamu je na obrázku 9.



Obr. 9: Grafické znázornění jednosměrně vázaného seznamu.

Jak je na obrázku 9 vidět, první prvek seznamu nese označení *first* a poslední prvek seznamu označení *last*. Tyto ukazatele slouží pouze pro potřeby programátora seznamu a nejsou navenek viditelné.

Pro základní práci se seznamem je implementováno několik nezbytných metod:

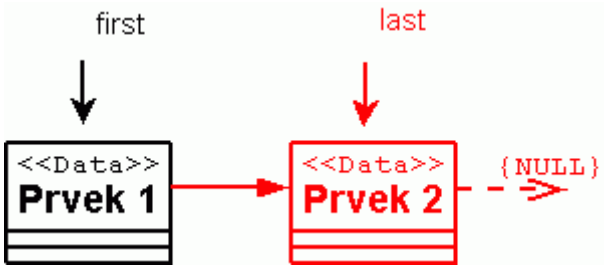
- Vložení prvku
- Nalezení prvku

- Smazání prvku
- Zjištění délky seznamu

4.2.1 Vložení prvku

Při vkládání prvku do seznamu hraje důležitou roli, zda požadujeme, aby byl seznam seřazen pro snadnější vyhledávání. Pokud tomu tak je, tak je nutné nejdříve vyhledat správnou pozici a poté prvek vložit. Zde je důležité zajistit, aby vložení proběhlo korektně a nedošlo k poškození integrity seznamu.

Jednodušší a častěji používanou variantou je vložení prvku na konec seznamu. Oproti předchozímu řešení je tato operace provedena v konstantním čase. Nevýhodou tohoto tipu vkládání prků je množství času potřebného pro vyhledávání. Vložení prvku na konec seznamu je znázorněno na obrázku 10.



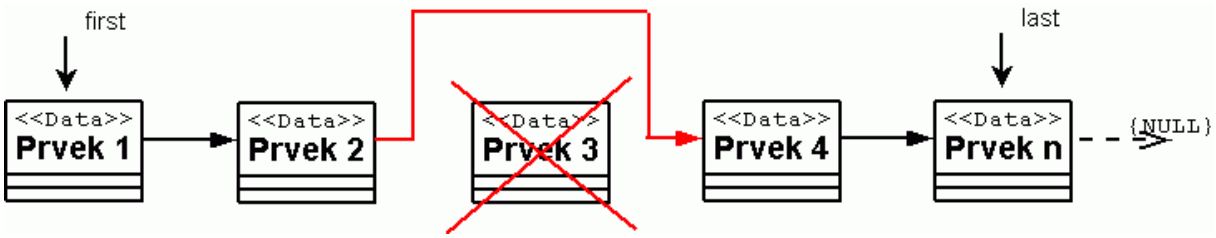
Obr. 10: Grafické znázornění vkládání prvku do jednosměrně vázaného seznamu.

4.2.2 Nalezení prvku

Nalezení prvku v jednosměrně vázaném seznamu spočívá v průchodu celým seznamem a hledání shody mezi hodnotou hledaného prvku a jednotlivými prvky seznamu. Časová složitost je závislá na velikosti seznamu, ale i na tom, zda je seznam seřazený či nikoliv.

4.2.3 Smazání prvku

Vymazání prvku je u tohoto typu seznamu nejsložitější úlohou jelikož je nutné zachovat integritu seznamu. Proto je zapotřebí nejprve vyhledat předchozí prvek a jemu předat ukazatel na prvek následující za mazaným. Tato operace je časově i implementačně nejnáročnější. Na obrázku 11 je graficky znázorněno odebrání prvku.



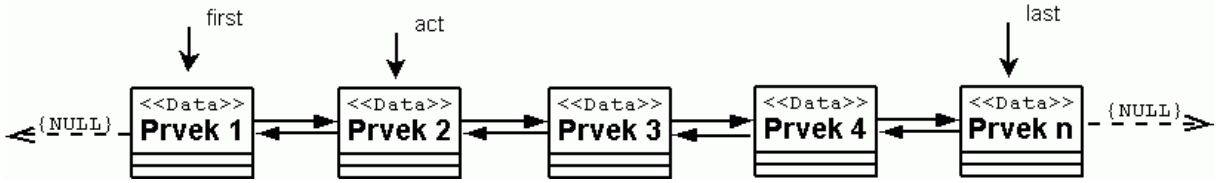
Obr. 11: Grafické znázornění odebírání prvku jednosměrně vázaného seznamu

4.2.4 *Zjištění délky seznamu*

Poslední metodou implementovanou u jednosměrně vázaných seznamů je zjištění velikosti seznamu. To spočívá v průchodu celým seznamem od začátku do konce. Při průchodu přes jednotlivé prvky se vždy jen počítadlo prvků zvýší o jedničku.

4.3 *Obousměrně vázaný seznam*

Obousměrně vázaný seznam je jakýmsi rozšířením jednosměrně vázaného seznamu. Abstraktní datový typ obousměrně vázaný seznam je v literatuře označován i jako větev neorientovaného grafu, protože je možné jí procházet oběma směry. První prvek seznamu, stejně jako poslední, nikam neukazuje, nebo-li ukazuje na NULL což je rozdíl oproti jednosměrně vázaného seznamu. Příklad obousměrně vázaného seznamu je na obrázku 12.



Obr. 12: Grafické znázornění obousměrně vázaného seznamu.

Zvláštností obousměrně vázaného seznamu je označení prvku *act*, který slouží pro interní potřeby seznamu.

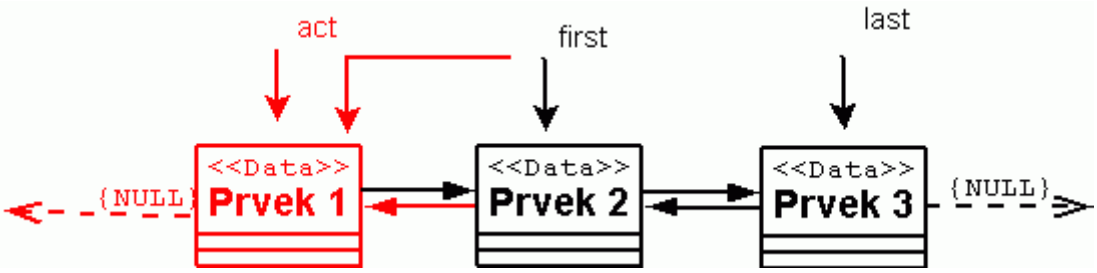
Obousměrně vázaný seznam implementuje stejné metody jako jednosměrně vázaný seznam, ale způsob implementace je jednodušší:

- Vložení prvku
- Nalezení prvku
- Smazání prvku
- Zjištění délky seznamu

4.3.1 *Vložení prvku*

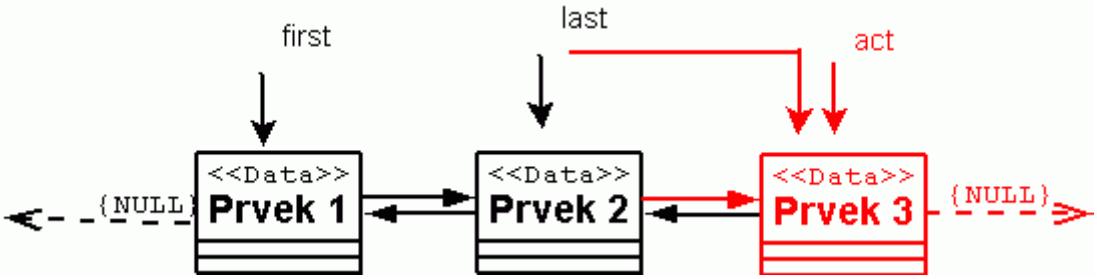
Vkládání prvku do obousměrně vázaného seznamu má několik možností. Záleží na tom, kam je nový prvek seznamu vkládán. Může se jednat o vkládání na začátek, na konec nebo na nějakou jinou pozici do seznamu.

Při vkládání prvku na začátek seznamu je důležité dát pozor na ukazatele *act* a *first*. Grafické znázornění vkládání prvku na začátek seznamu je na obrázku 13.



Obr. 13: Grafické znázornění vkládání prvku do obousměrně vázaného seznamu na první pozici.

Vkládání prvku na začátek je v čase konstantní stejně jako vkládání prvku na konec. Následující obrázek graficky znázorňuje vkládání prvku na konec.



Obr. 14: Grafické znázornění vkládání prvku do obousměrně vázaného seznamu na poslední pozici.

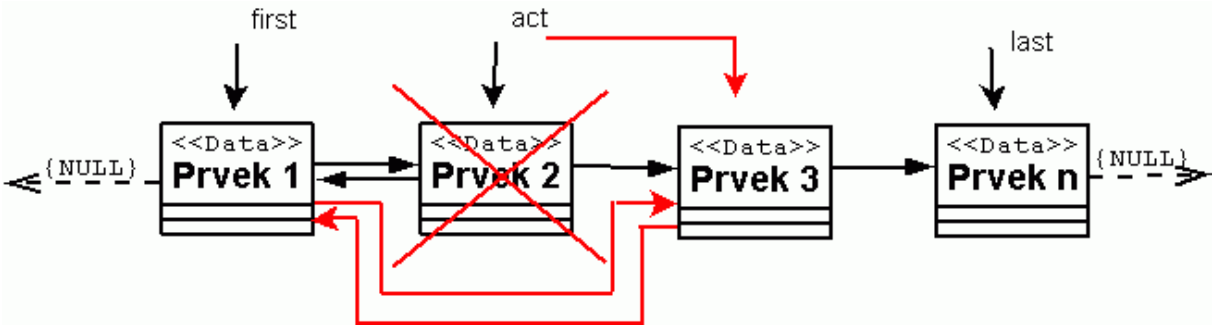
Poslední a nejnáročnější operací vkládání do obousměrně vázaného seznamu je vkládání do seznamu kamkoliv mezi první a poslední pozici. Tato možnost se používá pro seznamy, které jsou tříděny již při jejich vytváření. Jedná se o časově složitou operaci, jelikož se musí najít pozice, na kterou bude prvek vložen.

4.3.2 Nalezení prvku

Nalezení prvku v obousměrně vázaném seznamu spočívá v průchodu seznamem a hledání shody mezi hledaným prvkem a jednotlivými prvky seznamu. Časová složitost je závislá na velikosti seznamu, ale i na tom, zda je seznam seřazený či nikoliv. Jelikož se jedná o obousměrně vázaný seznam je jedno jakým směrem se bude seznamem pohybovat.

4.3.3 Smazání prvku

Vymazání prvku je hlavním důvodem rozšíření jednosměrně vázaného seznamu na obousměrný. Při mazání prvku je nutné znát předchozí a následující prvek, což je jednoduché, protože každý prvek obsahuje ukazatel na předchozí a následující prvky seznamu. Grafické znázornění odebírání prvku z obousměrně vázaného seznamu je na obrázku 15.



Obr. 15: Grafické znázornění odebírání prvku obousměrně vázaného seznamu

4.3.4 Zjištění délky seznamu

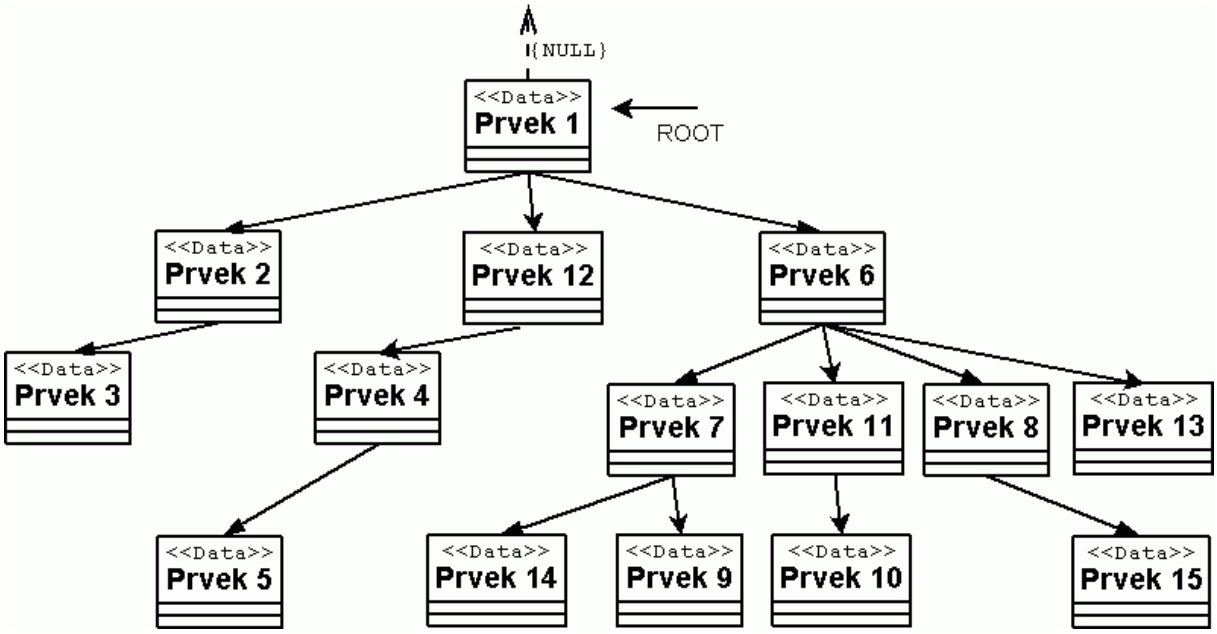
Stejně jako u jednosměrně vázaných seznamů je i zde implementována metoda pro zjištění velikosti seznamu. Celý seznam je procházen od začátku na konec nebo obráceně a je spočítán počet prvků seznamu.

4.4 Stromy

Strom reprezentuje abstraktní datový typ a můžeme ho chápat jako zobecněný seznam, kde může mít každý prvek jednoho, ale i více následníků. Při práci s abstraktní datovou strukturou jakou je strom je nutné znát několik následujících pojmů:

- **Kořen stromu –** takto se označuje vrchol, který nemá žádného rodiče, ale pouze potomky. Kořen stromu bývá často označován anglickým výrazem *root*.
- **List stromu –** jako listy jsou označeny ve stromu prvky, které na rozdíl od kořene nemají žádné potomky.
- **Uzel stromu –** každý prvek, který má rodiče a jednoho či více potomků se potom označuje jako uzel stromu.
- **Podstrom –** každá část stromu, který rozvijí strom jako celek může být označena jako podstrom. Jedná se o entitu stromu, která by vznikla přerušením spojnice kořene stromu a kořene podstromu.
- **Hloubka stromu –** je dána vzdáleností mezi kořenem a nejvzdálenějším listem stromu. Někdy se pro hloubku stromu zavádí i pojem **výška stromu.**
- **Hladina stromu –** je určena počtem uzlů a listů stromu, které leží ve stejné hloubce.
- **Šířka stromu –** nejširší hladina stromu určuje šířku celého stromu.

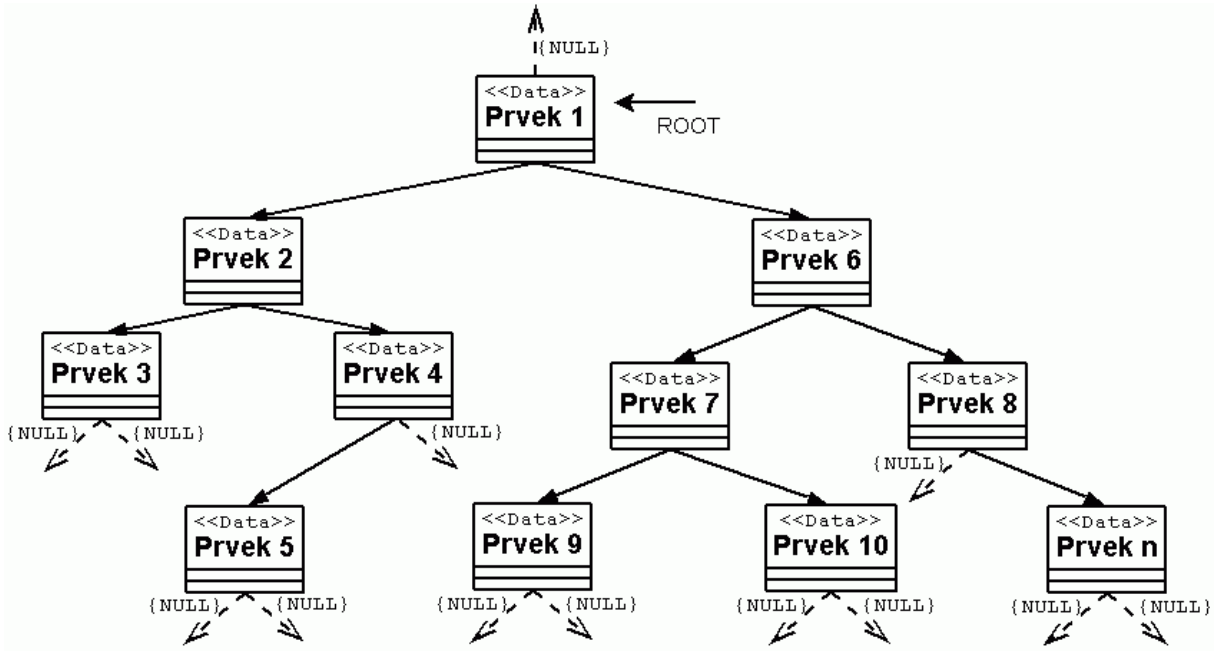
Stromy jsou obecně grafy orientované, ale orientace hran se zpravidla nekreslí. Orientace je dána vztahem rodič-potomek (nebo-li hrany jsou orientovány z vrcholu ležícího výše do vrcholu ležícího níže). Grafické znázornění stromu je na obrázku 16 (je zde naznačena i orientace hran stromu):



Obr. 16: Grafické znázornění stromu.

4.4.1 Binární strom

Binární strom je zvláštní případ stromu, kdy má každý uzel právě dva potomky. Tento konkrétní typ stromu se používá například pro vyhledávání nebo jako vnitřní struktura pro expertní systémy. Příklad obecného binárního stromu je na obrázku 17:



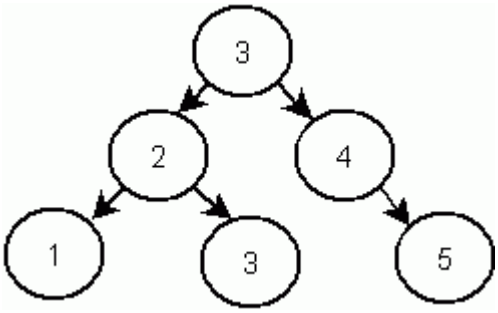
Obr. 17: Grafické znázornění binárního stromu.

4.4.2 Binární vyhledávací strom

Binárním vyhledávacím stromem se označuje binární strom, pro jehož vytváření a práci s ním platí určitá pravidla. Uzly tohoto stromu totiž obsahují klíče. Pro jednoduchost si je lze představit jako celá čísla. Tyto klíče potom slouží jako systém pro uspořádání stromu. Toto uspořádání je realizováno tak, že klíče levého podstromu jsou vždy menší než klíče pravého podstromu a nebo naopak.

Vyhledávat v takto uspořádaném systému klíčů je jednoduché a patrné z výše uvedených informací. Množina hodnot, ve které hledáme je vždy rozdělena na dvě části, které jsou určeny aktuálním uzlem. Časová složitost takového vyhledávání je závislá na velikosti levého a pravého podstromu, který vznikne při průchodu kořenem tohoto podstromu.

Příklad binárního vyhledávacího stromu je na obrázku 18. Klíče levého podstromu jsou vždy menší nebo rovny jak hodnoty pravého podstromu.



Obr. 18: Náhodně vygenerovaný binární vyhledávací strom

Pro potřeby práce s binárním stromem je nutné implementovat několik základních metod. Vkládání a hledání prvku bylo popsáno v předchozí části, a proto jedinou základní metodou, o které nebyla zmínka je mazání prvku ze stromové struktury.

Mazání prvků binárního vyhledávacího stromu je závislé na počtu potomků stromu nebo na pozici uzlu ve stromu. Rozlišujeme tyto případy:

- Mazaný uzel má dva potomky
- Mazaný uzel má jednoho potomka
- Mazaný uzel je bez potomků

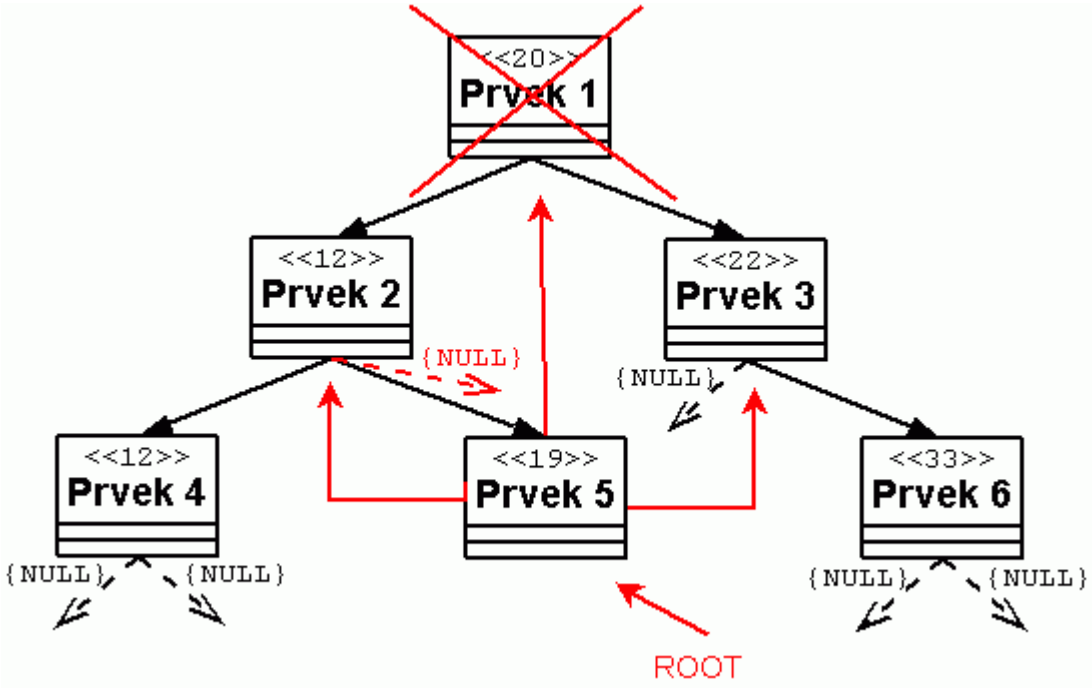
4.4.2.1 Mazání uzlu se dvěma potomky

Tato varianta mazání prvku má dvě různé možnosti řešení: levou a pravou:

Levá varianta – v tomto případě je nalezen nejpravější prvek levého podstromu, který je určen mazaným prvkem a tento prvek je poté nahrazen za mazaný uzel.

Pravá varianta – oproti levé variantě je zrcadlově otočená. Je nalezen nejlevější prvek pravého podstromu, který je určen mazaným prvkem a tento prvek je poté nahrazen za mazaný uzel.

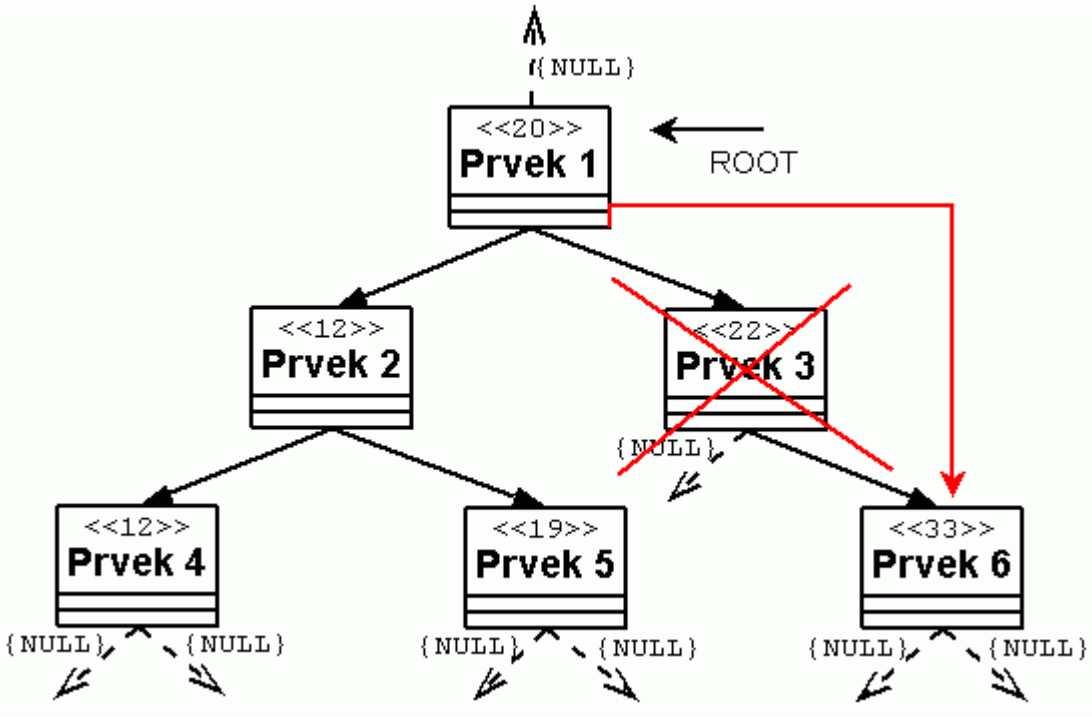
Na následujícím obrázku je zobrazena levá varianta, kdy mazaným prvkem je sám kořen stromu. Zde dojde navíc jen k předání ukazatele *root*, který na kořen ukazuje.



Obr. 19: Mazání prvku binárního vyhledávacího stromu se dvěma potomky

4.4.2.2 Mazání uzlu s jedním potomkem

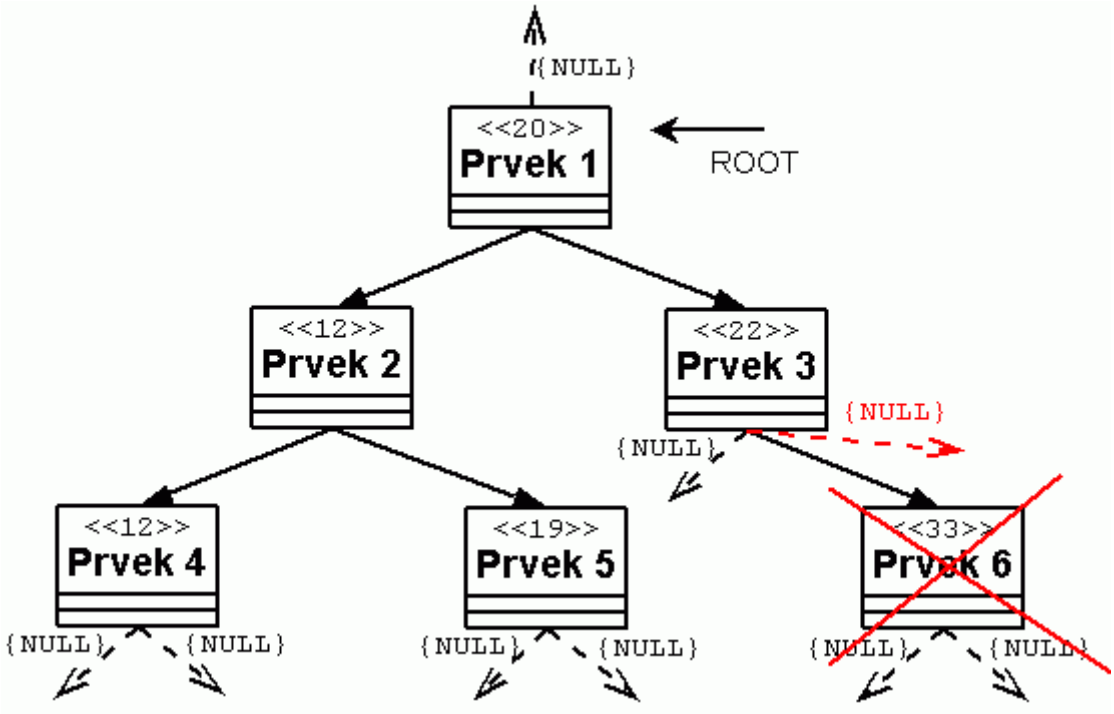
Oproti předchozímu řešení je tato úloha mnohem jednodušší. Mazaný prvek se pouze nahradí jeho jediným potomkem. Mazání prvku s jedním potomkem je graficky znázorněno na obrázku 20.



Obr. 20: Mazání prvku binárního vyhledávacího stromu s jedním potomkem

4.4.2.3 Mazání uzlu bez potomků

Nejjednodušší variantou mazání uzlu je případ, kdy je uzel listem stromu a tudíž nemá žádné potomky. Zde je prvek stromu prostě smazán jak ukazuje obrázek 21.



Obr. 21: Mazání prvku binárního vyhledávacího stromu bez potomka

4.5 Vyhledávání v grafech

Jednou z nejdůležitějších úloh, které jsou v grafech velmi často prováděny, je vyhledávání nebo změna některých prvků grafu. Pokud však chceme s prvkem nějakým způsobem pracovat, je třeba tento prvek nejprve najít. Dalším důvodem pro implementaci metod průchodu grafem je zjištění dostupnosti prvků grafu z výchozího bodu. Tato úloha by se sice dala vyřešit prostým převedením grafu do grafické podoby a vyhledáním, ale pokud by byl graf velmi rozsáhlý, je jednodušší užít některý ze systematických postupů při vyhledávání. Proto vzniklo několik algoritmů, jenž se v zásadě dělí na 2 skupiny. První skupinou jsou algoritmy, procházející graf náhodně bez jakékoliv snahy odhadnout ideální cestu grafem. Druhá skupina algoritmů přistupuje k řešení problému vyhledávání v grafech mnohem efektivnějším způsobem a k vyhledávání se využívá informovaných metod, díky kterým je možné nalézt optimálnější řešení.

4.5.1 Slepé prohledávání

Tato rodina algoritmů využívá postupného procházení celého grafu. Z pravidla nebývá implementována žádná funkce, která hodnotí průchod přes jednotlivé prvky a přiřazuje jim prioritu. Tyto priority by potom sloužily k efektivnějšímu nalezení cesty k cíli.

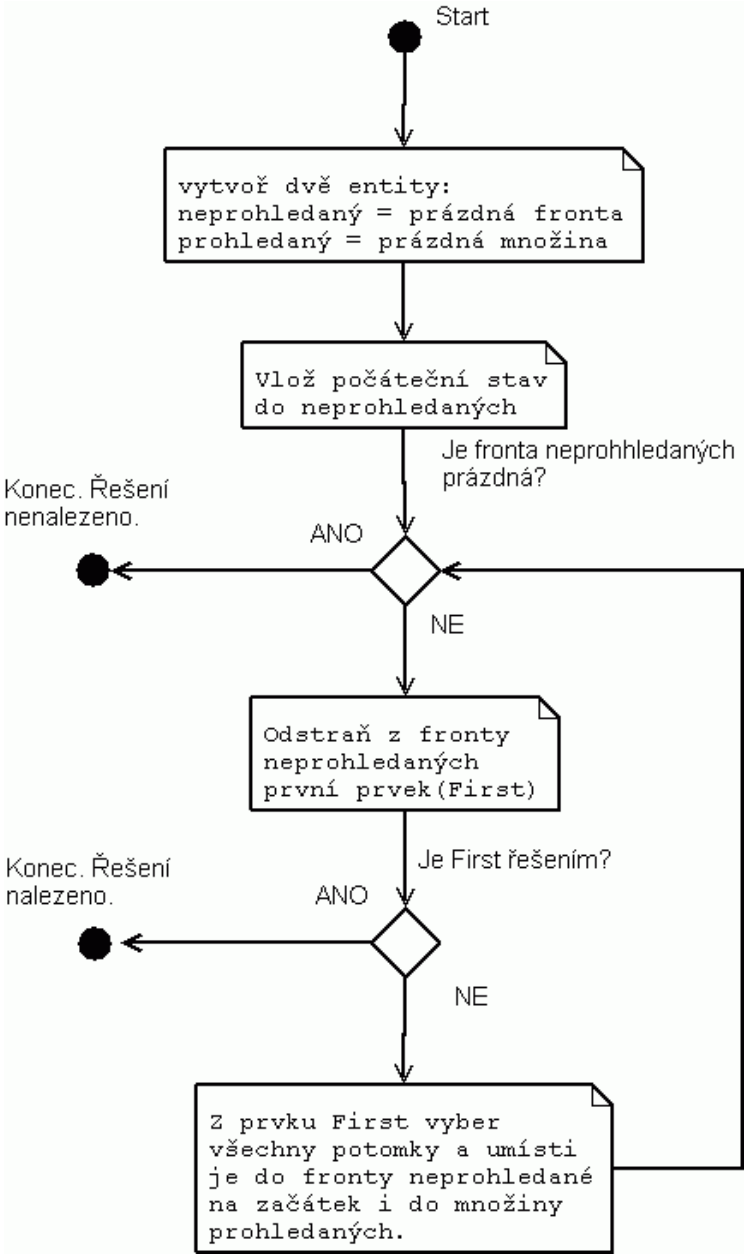
4.5.1.1 Prohledávání do hloubky

Prohledávání do hloubky, někdy označovanému jako *DFS* (Depth First Search), je jedním ze dvou hlavních reprezentantů slepého prohledávání grafů. Princip činnosti tohoto algoritmu je v upřednostnění uzlů grafu, jež mají od počátku největší vzdálenost. Princip funkce metody by se dal shrnout do následujících kroků:

- Na začátku je programem vytvořena prázdná množina pro prvky, které již byly prohledány. Současně s tím je vytvořena prázdná fronta pro prvky, které ještě prohledány nebyly.
- Výchozí vrchol je vložen do fronty pro neprohledané vrcholy
- Proběhne test na prázdnost fronty neprohledaných. Tento test slouží pro ukončení v případě nenalezení výsledku.
- V dalším kroku je z fronty neprohledaných vybrán první prvek a označen jako *first*.
- Vrchol označený jako *first* je testován na shodnost s vyhledávaným prvkem.

- Do fronty neprohledaných se vloží všechny následující vrcholy na začátek. Ty se kontrolují a vkládají i do množiny prohledaných, aby nedocházelo k zacyklení kvůli testu stejných prvků vícekrát.

Na tomto algoritmu je zajímavá právě práce s frontou neprohledaných vrcholů. Tato fronta má povahu zásobníku a prvky jsou vkládány na jeho vrchol tak, aby byl vybrán z fronty prvek, který je ve frontě nejkratší dobu. Prvky jsou tedy z fronty vybírány od nejvzdálenějšího vrcholu k vrcholu výchozímu. Na obrázku 22 je vývojový diagram algoritmu prohledávání do šířky.



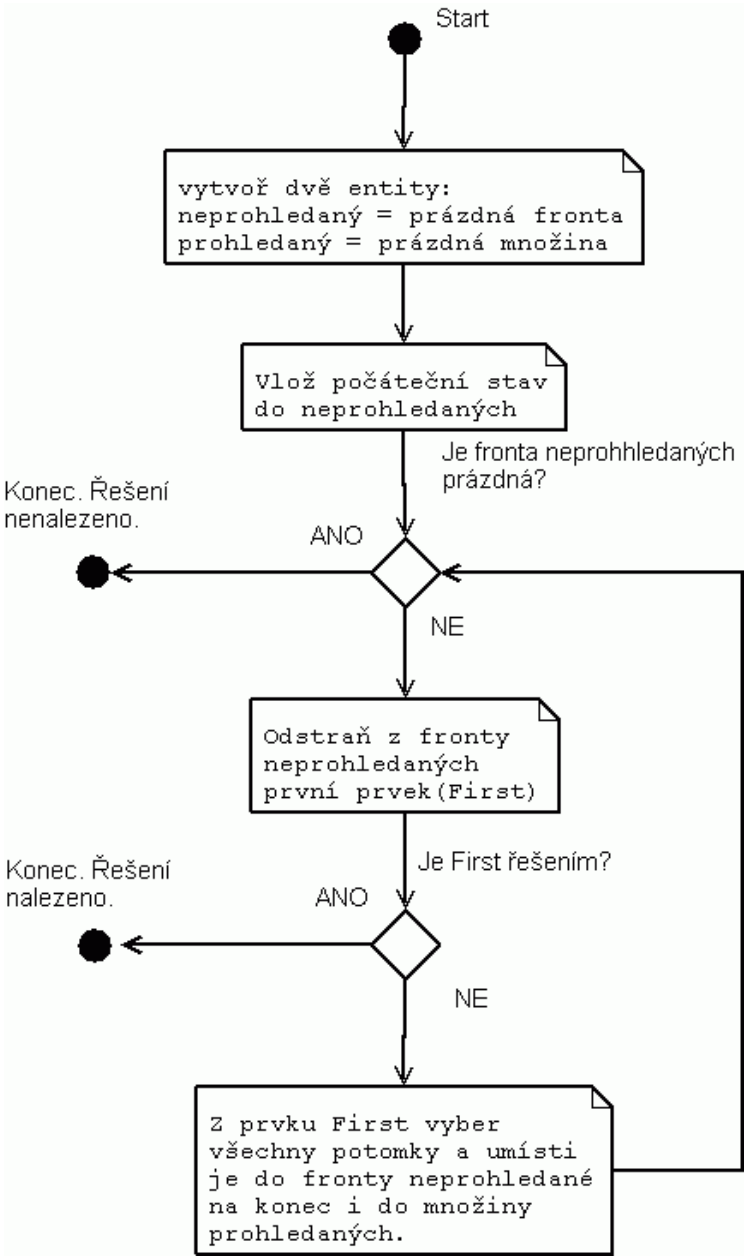
Obr. 22: Vývojový diagram pro algoritmus prohledávání do hloubky.

4.5.1.2 Prohledávání do šířky

Tento algoritmus bývá označován jako algoritmus *BFS* (Breadth First Search) . Prohledávání grafu do šířky lze jednoduše charakterizovat jako záplavu při hledání. Hledání totiž probíhá tak, že u výchozího vrcholu prohledáme potomky. Pokud není nalezena hodnota nebo není projitý celý graf, tak se celý postup neustále opakuje. Tato jednoduchá metoda dává nejlepší výsledky při hledání nejkratších cest v grafech bez ohodnocených přechodů mezi uzly. Vývojový diagram pro algoritmus prohledávání do šířky je znázorněn na obrázku 23. Celý postup vyhledávání by se dal shrnout do následujících bodů:

- Na začátku je programem vytvořena prázdná množina pro prvky, které již byly prohledány. Současně s tím je vytvořena prázdná fronta pro prvky, které ještě prohledány nebyly.
- Výchozí vrchol je vložen do fronty pro neprohledané vrcholy
- Proběhne test na prázdnost fronty neprohledaných. Tento test slouží pro ukončení v případě nenalezení výsledku.
- V dalším kroku je z fronty neprohledaných vybrán první prvek a označen jako *first*.
- Vrchol označený jako *first* je testován na shodnost s vyhledávaným prvkem.
- Do fronty neprohledaných se vloží všechny následující vrcholy na konec. Ty se kontrolují a vkládají i do množiny prohledaných, aby nedocházelo k zacyklení kvůli testu stejných prvků vícekrát.

Na tomto algoritmu je podstatné to, že prvky jsou do fronty neprohledaných vkládány na konec a tak se z fronty vybere vždy ten prvek, který je ve frontě nejdéle. Proto dostal algoritmus název prohledávání do šířky.



Obr. 23: Vývojový diagram pro algoritmus prohledávání do šířky.

4.5.1.3 Prohledávání do hloubky s omezenou hloubkou

Algoritmus *DLS* (Depth Limited Search) vznikl jako úprava algoritmu vyhledávání do hloubky jak napovídá samotný název. Princip činnosti je tedy totožný s algoritmem prohledávání do šířky. Odlišnost je pouze v hlídané hladině zanoření. Tato varianta vyhledávání se s výhodou užívá v situacích, kdy je známá hloubka vnoření vyhledávaného prvku oproti prvku výchozímu.

4.5.1.4 Iterativní prohledávání do hloubky s omezenou hloubkou

Tento algoritmus nese označení *IDLS*. Hlavní nevýhodou prohledávání s omezenou hloubkou je neschopnost nalézt prvek, pokud leží na vzdálenější hladině než je hladina odhadovaná. V tomto případě je hloubka vnoření, při nenalezení hledaného prvku, inkrementována a celý postup hledání se opakuje dokud nedojde k úspěšnému nalezení nebo k projití celého grafu.

4.5.2 Informované metody prohledávání

Tyto metody jsou oproti slepému prohledávání vybaveny funkcemi, které umožňují řídit průchod grafem, a tak dojít od zdroje k cíli v mnohem kratším čase. Jako nástroj pro řízení průchodu slouží ohodnocování všech stavů, což vede ke zvýhodnění některých cest oproti jiným. Nevýhodou tohoto řešení je právě nutnost implementace nástroje pro ohodnocování stavů.

4.5.2.1 Best FS

Best FS je algoritmus postavený na vyhledávání do šířky. Rozšířením je mechanismus rozhodování o tom, zda se konkrétně zpracovávaný uzel grafu blíží k řešení více než ostatní.

Implementace algoritmu potom vyžaduje pouze použití prioritní fronty namísto normální. Prvky jsou tedy vkládány do fronty s přidělenými prioritami a podle nich jsou také z fronty odebírány. Přidělování priorit se nedá obecně nijak stanovit a proto je třeba upravit řešení přímo pro ten který případ.

*4.5.2.2 A **

Tento algoritmus je pouhým rozšířením předchozího o možnost přidělování priorit nejen podle stavů, které se zdají být nejblíže řešení, ale i podle již prošlých cest, které nevedly k řešení nalezení cesty.

4.5.3 Porovnání vyhledávacích algoritmů

Hlavním úkolem vyhledávacích algoritmů je určení dostupnosti prvků grafu z výchozího bodu. To může být vhodné například při určování spojitosti grafu. Pro tyto účely bylo vyvinuto několik algoritmů, které však více nebo méně vycházejí z algoritmu prohledávání do šířky nebo do hloubky.

Prohledávání do šířky je implementačně nenáročný algoritmus stejně jako prohledávání do hloubky. Hlavní síla této metody určování dostupnosti prvků v grafu je vybírání těch uzlů, které se nacházejí v bezprostředně blízkém okolí výchozího prvku. Tento postup se používá i u algoritmů pro určování nejkratší cesty v grafech. Proto je vyhledávání do šířky užíváno pro hledání nejkratší cesty v neohodnocených grafech.

Prohledávání grafu do hloubky je naopak využíváno pro vyhledávání v krajních oblastech grafu. Je možné jej využít v situacích hledání extrémů grafu, kde by použití vyhledávání do šířky nebylo vhodné, zejména pro délku doby výpočtu.

Obě dvě metody je možné kombinovat nebo výpočet provádět současně. V případě kombinace dochází k volbě toho kterého algoritmu na základě odhadu vzdálenosti vyhledávaného prvku. Současné použití obou metod potom, při zdvojnásobení nároků na výpočetní kapacitu, umožňuje vyhledávat bez odhadu.

..

4.6 Hledání nejkratší cesty

Dalším z hlavních problémů teorie grafů je způsob, jakým lze grafy procházet (viz. předchozí kapitola). Pokud však hrany spojující dva vrcholy označíme číselně podle nějakého kritéria, vznikne graf ohodnocený. Nad takovouto množinou grafů můžeme provádět vyhledávání nejkratší cesty. V praxi se tyto výpočty používají u *link-state* protokolů, které slouží ke směrování paketových dat v IP prostoru. Ohodnocení hran je zde zastoupeno ohodnocením jednotlivých přenosových cest mezi směrovači v závislosti na jejich průchodnosti.

4.6.1 Neohodnocený graf

Úlohu vyhledávání nejkratší cesty lze aplikovat i na grafy neohodnocené. Délka nejkratší cesty je potom počítána jako počet hran, které vedou z počátečního do koncového bodu. Tato vzdálenost by odpovídala vzdálenosti v ohodnoceném grafu, kde by každá hrana byla ohodnocena jedničkou.

Jako algoritmus pro vyhledávání nejkratší cesty v neohodnocených grafech je využíváno, v kapitole 4.5.1.2 popsaného algoritmu *BFS* (Breadth First Search), který je označován jako algoritmus procházení do šířky. Jiné označení tohoto postupu využívaného pro nalezení nejkratší cesty v neohodnocených grafech je algoritmus vlny. Průchod grafem si totiž můžeme představit jako vlny, které se šíří z počátečního bodu ke koncovému.

4.6.2 Ohodnocený graf

Většina aplikací zabývající se vyhledáváním nejkratší cesty však nepracuje s jednotkově ohodnoceným grafem. Jakým způsobem bude graf ohodnocen určuje situace, ve které je této struktury využíváno. V zásadě se ale tyto aplikace dělí na dvě velké skupiny a to podle toho, jaké ohodnocení je v grafu použito.

- Nezáporně ohodnocené grafy
- Obecně ohodnocené grafy

4.6.2.1 Nezáporně ohodnocený graf

Nezáporně ohodnocené grafy pro ohodnocení hran využívají nezáporná celá čísla. Tím se problém vyhledávání nejkratší cesty stává složitější úlohou. Algoritmus *BFS* by si zde nevystačil, jelikož prohledává celou oblast grafu v záplavách. Proto bylo nutné implementovat několik algoritmů, které k problému přistupují efektivněji. Mezi tyto algoritmy patří zejména:

- Dijkstrův algoritmus
- Floyd-Warhallův algoritmus

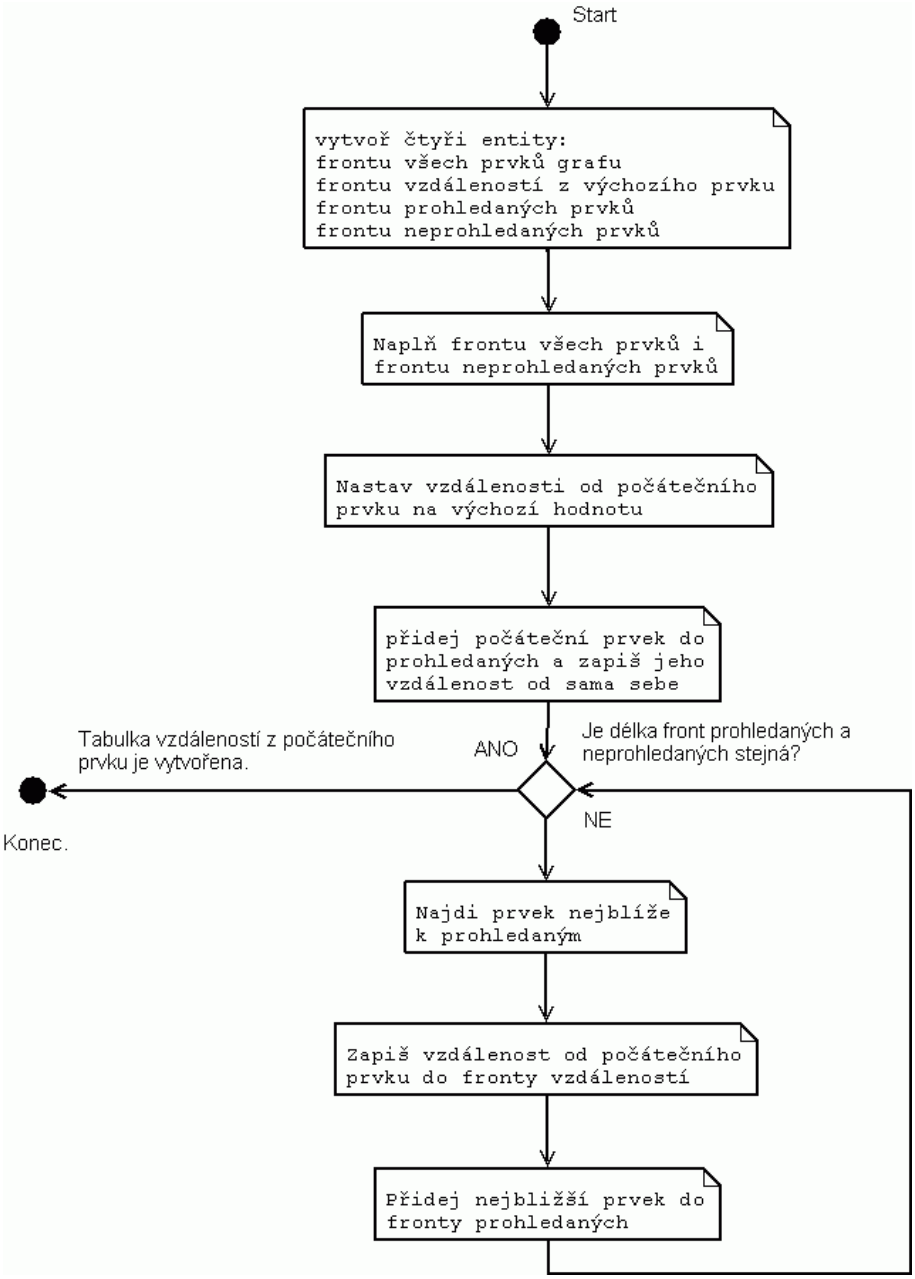
4.6.2.1.1 Dijkstrův algoritmus

Dijkstrův algoritmus vyhledává nejkratší cesty z výchozího vrcholu do ostatních v nezáporně ohodnoceném neorientovaném grafu. Podstatu práce Dijkstrova algoritmu lze shrnout do několika základních bodů:

- Na začátku je programem vytvořeno několik front. Fronta pro prohledané a pro neprohledané prvky. Dále je vytvořena fronta všech prvků grafu a s touto frontou korespondující seznam vzdáleností z výchozího vrcholu.
- Fronta neprohledaných a všech prvků je naplněna současně.
- Z fronty prohledaných prvků je vybrán počáteční prvek a je umístěn mezi prohledané prvky a zároveň je zapsána jeho vzdálenost od výchozího (jde o vzdálenost prvku od sama sebe, což je pochopitelně 0).
- Postupně se prohledávají další prvky grafu. Vždy se vybere ten prvek, který je k výchozímu nejblíže, umístí se mezi prohledané a zaznamená se jeho vzdálenost.

- Celý postup se opakuje, dokud nejsou projity všechny prvky grafu.

Algoritmus je konečný, protože v každém kroku se zmenšuje množina neprohledaných prvků právě o jeden. Vývojový diagram pro Dijkstrův algoritmus je na obrázku 24. Tento algoritmus pracuje, jak již bylo uvedeno, s neohodnoceným grafem a proto je nutné pro uložení grafu v paměti zvolit vhodnou datovou strukturu reprezentující graf. Jako nejvýhodnější se z hlediska návrhu jeví využít seznamu hran.



Obr. 24: Vývojový diagram Dikstrova algoritmu.

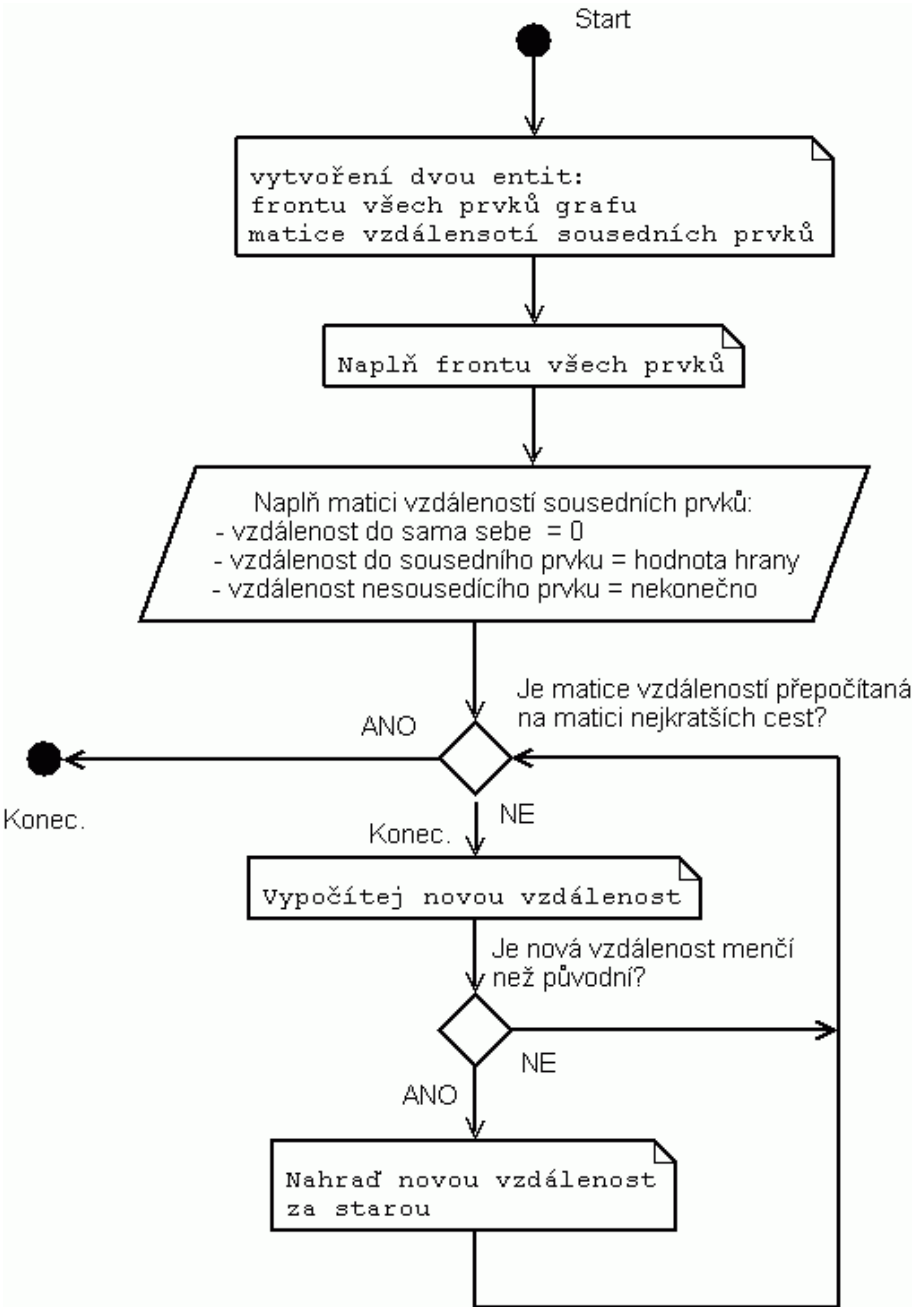
4.6.2.1.2 Floyd-Warshallův algoritmus

Oproti Dijkstrovu algoritmu pracuje s orientovaným grafem s nezáporně ohodnocenými hranami. Díky práci pouze s orientovanými hranami dokáže vypočítat minimální vzdálenost mezi libovolnou dvojicí prvků grafu. Pokud bychom tohoto chtěli dosáhnout za pomoci předchozího algoritmu, bylo by potřeba provést tento postup tolikrát, kolik je prvků v grafu a to by u rozsáhlých grafů bylo velmi zdlouhavé a náročné na výpočetní výkon.

Základní podmínkou Floyd-Warshallova postupu výpočtu nejkratší cesty grafem je označení jednotlivých vrcholů přirozenými čísly. Největší síla je v tom, že vzdálenosti se nepočítají přímo, ale v iteracích jejichž počet je roven počtu prvků grafu. Postup výpočtu nejkratších cest by se dal shrnout takto:

- Vytvoř frontu všech vrcholů a matici vzdáleností, která vznikne rozšířením fronty vzdáleností z jednorozměrného na dvojrozměrné pole.
- Naplň frontu všech prvků a přiřaď jim číselné hodnoty od jedné.
- Naplň matici vzdáleností podle následujících pravidel:
    - Vzdálenost prvku sama od sebe je 0 (diagonála matice bude nulová).
    - Vzdálenost od prvku k sousednímu je rovna nezápornému ohodnocení hrany.
    - Vzdálenost z prvku do nesousedícího je rovna nekonečnu.
- Dokud nebude matice vzdáleností projita celá, provádí se jednotlivé iterace. V každé iteraci je projita celá matice vzdáleností a hledá se kratší cesta než je cesta nastavená.

Tento algoritmus je velmi jednoduchý a efektivní. Pokud bychom si chtěli u každého prvku pamatovat, jak se do něho dostaneme, museli bychom do algoritmu implementovat mechanismus zachytávání projitých uzlů, což by vyžadovalo vyšší paměťové nároky. Oproti Dijkstrovu algoritmu se jedná o mnohem efektivnější postup výpočtu nejkratší cesty, ale nelze ho použít na neorientovaný graf. V praxi se mnohem častěji využívá práce s neorientovanými grafy, a tak je Floyd-Warshallův algoritmus využíván méně častěji.



Obr. 25: Vývojový diagram Floyd-Warshallova algoritmu

4.6.2.2 Obecně ohodnocený graf

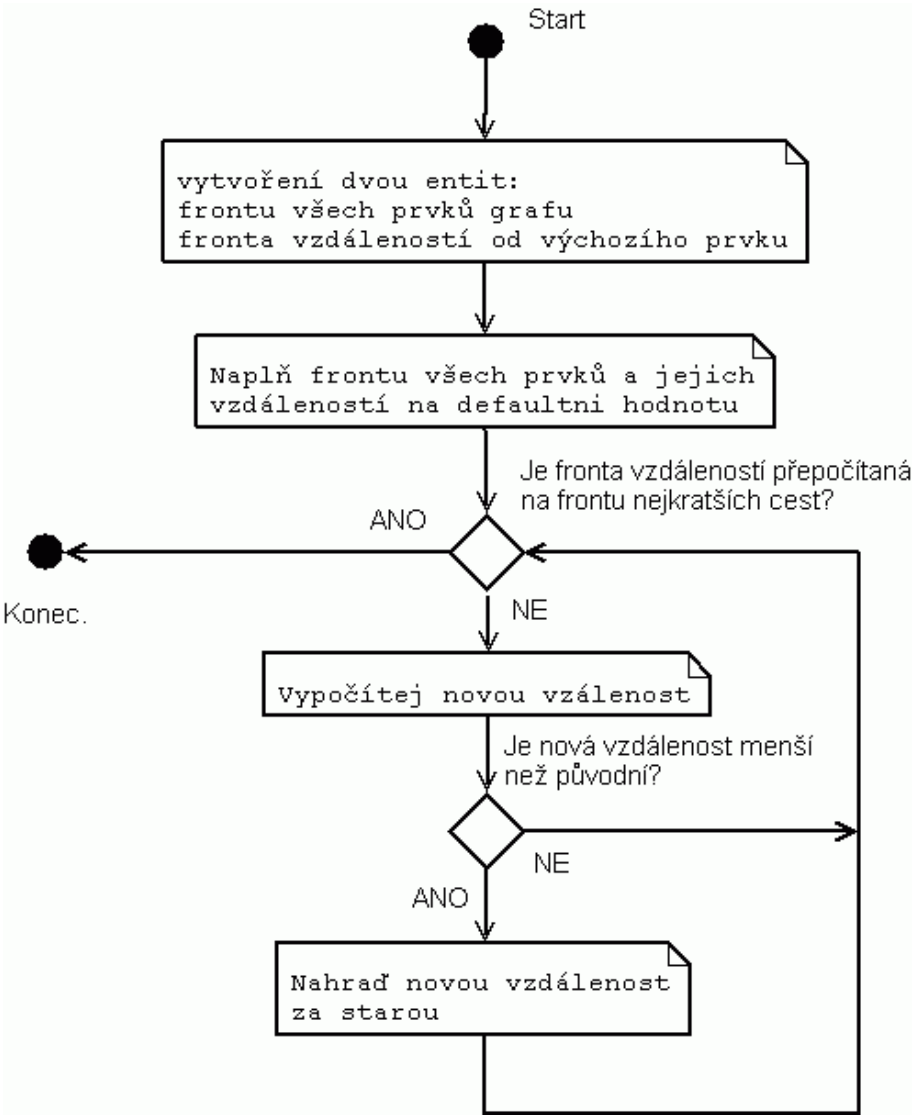
Obecně ohodnocený graf není úplným opakem nezáporně ohodnoceného grafu, protože může obsahovat jak kladné, tak i záporně hodnocené hrany. Kdyby kladné ohodnocení hrany znamenalo dobu, která je potřeba na projití přes hranu, tak by záporné číslo ukazovalo, že hranou se dá projít dříve než bychom jí vůbec projít chtěli. Takovéto označování hran a následná práce s nimi je velmi složitá a nejednoznačná, proto se v praxi obecně ohodnocené grafy v zásadě nepoužívají. Jediným zástupcem je tedy Bellman-Fordův algoritmus.

4.6.2.2.1 Bellman-Fordův algoritmus

Jak již bylo uvedeno, tento algoritmus vyhledá nejkratší cestu v obecně ohodnoceném grafu. Základním funkčním pilířem je metoda, která zajišťuje aktuálně nastavenou hodnotu nejkratší vzdálenosti. Vývojový diagram Bellman-Fordova algoritmu je na obrázku 27. Celý postup výpočtu je možné shrnout do následujících bodů:

- Vytvoř dvě základní entity pro výpočet: fontu všech prvků procházeného grafu a frontu vzdáleností od výchozího prvku.
- Naplň vytvořené fronty. Prvek fronty vzdálenosti reprezentující vzdálenost d počátku je nastaven na 0.
- Dokud nebude projita celá fronta uzlů, počítej v každé iteraci novou vzdálenost jednotlivých prvků a porovnej ji s uloženou hodnotou.

Za předpokladu, že nejkratší cesta neobsahuje hran tolik, kolik je hran grafu, můžeme běh algoritmu urychlit nastavením počtu iterací právě na hodnotu počtu hran, které jsou nejkratší cestou k výchozímu bodu.



Obr. 26: Vývojový diagram Bellman-Fordova algoritmu.

4.6.3 Porovnání metod pro výpočet minimální cesty

Pro vyhledávání v grafech bylo vyvinuto spoustu algoritmů a postupů. Při výběru nejvhodnější metody pro nalezení nejkratší cesty záleží na reprezentaci, orientaci nebo na způsobu ohodnocení grafu.

Jako nejuniverzálnější se jeví Dijkstrův algoritmus, jelikož dokáže prohledávat grafy bez ohledu na orientaci. V tomto je jeho největší síla. Oproti ostatním algoritmům pokulhává ve výpočtech minimálních cest pro všechny prvky grafu. Pokud bychom tedy chtěli určit minimální vzdálenosti pro všechny prvky grafu, bylo by nutné provést výpočet tolikrát, kolik je prvků v grafu. Tento nedostatek je vykoupen výše zmíněnou nezávislostí na orientaci v grafu a Dijkstův algoritmus je v praxi nejužívanějším postupem pro výpočet minimální vzdálenosti.

Floyd-Warshallův algoritmus je oproti Dijkstrovu algoritmu mnohem jednodušší a je možné najednou vypočíst vzdálenosti pro všechny prvky grafu. Výrazným omezením je ale neschopnost pracovat s neorientovanými grafy. Částečným řešením je vypočíst hodnotu minimální vzdálenosti pro původní a pozměněnou variantu grafu. Výsledky ale nebudou totožné. Dostaneme potom jinou nejkratší cestu z výchozího prvku do vyhledávaného než tomu je u cesty z vyhledávaného do výchozího.

Nejméně užívanou variantou ohodnocení grafů, jsou obecně ohodnocené grafy. Výpočet minimální vzdálenosti je potom složitou a nejednoznačnou úlohou. Zástupcem určování minimální cesty v obecně ohodnocených grafech je Bellman-Fordův algoritmus. Oproti předchozím algoritmům obsahuje mechanismy pro detekci záporných cyklů, kterých není u nezáporně ohodnocených grafů zapotřebí.

4.7 Kostry grafu

Kostrou grafu nazýváme každý libovolný pod-graf, který vznikne z původního. Kostra grafu obsahuje všechny uzly původního grafu, ale jen několik nejdůležitějších hran. Vybrány jsou jen ty, které spojují všechny uzly. Z uvedených informací je jasné, že graf může mít více než jednu kostru, pokud existuje dostatečné množství hran, nebo pokud již původní graf není kostrou.

Všechny kostry neohodnoceného grafu jsou různé, ale navzájem rovnocenné. Mají stejný počet uzlů a hran. Pokud ale budeme chtít vyhledávat kostry v ohodnoceném grafu, budou kostry rozdílné v součtu ohodnocení hran. Potom můžeme zavést pojem minimální kostra grafu. Každý graf může mít, stejně jako koster, několik minimálních koster.

Hledání minimálních koster v grafu je velmi užívaným aparátem v praktických aplikacích. Proto existuje několik navrhovaných řešení, kterými se zabývá vědní disciplína s názvem teoretická informatika. Tato věda se však zabývá pouze neorientovanými grafy.

S pojmem minimální kostra grafu souvisí i pojem počet minimálních koster v grafu. Díky různému ohodnocení hran může vzniknout několik minimálních koster, které budou navzájem rozdílné. Všechny tyto kostry jsou minimálními a jejich počet lze stanovit z matice sousednosti nebo z matice incidence, které graf popisují.

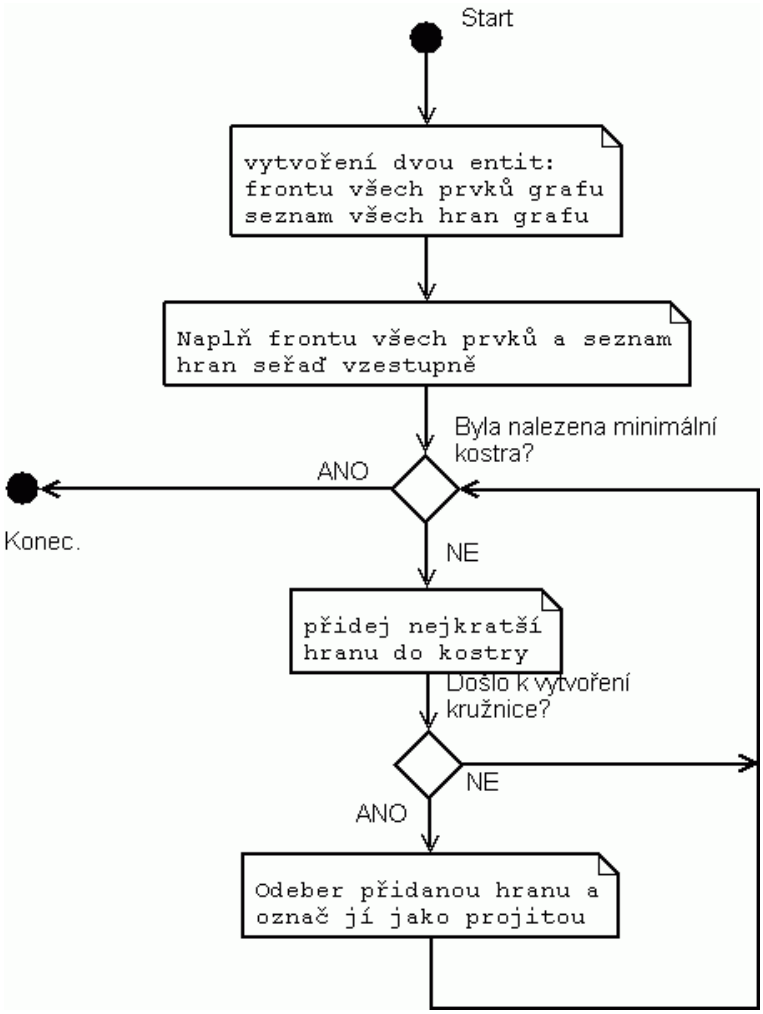
Obecně lze spočítat kostry vycházející z různých prvků a vzájemně je porovnat. Pro to neexistuje žádný předem stanovený postup, ale podle Cayleyho formule [11] lze stanovit, že

graf obsahující n vrcholů může obsahovat až n^{n-2} koster. Toto pravidlo platí pro všechny grafy, jejichž počet uzlů je větší nebo roven třem.

4.7.1 Kruskalův algoritmus

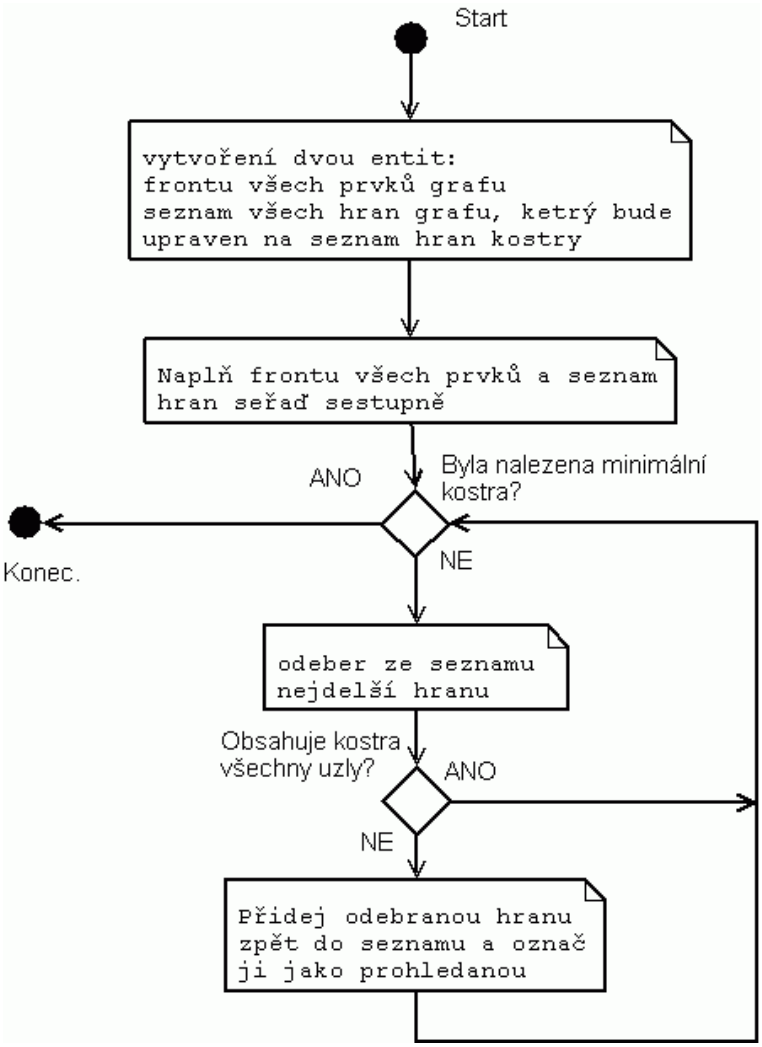
Tento postup nalezení minimální kostry grafu bývá také někdy označován jako Kruskalův hladový algoritmus. Na počátku je vytvořen seznam hran a ty jsou postupně odebírány nebo je vytvořen prázdný seznam, jež je postupně plněn. Nalezení minimální kostry lze tedy realizovat dvěma způsoby:

- **Postupné přidávání hran**
    - Na začátku je vytvořen prázdný seznam hran, který slouží jako datová struktura pro minimální kostru grafu.
    - Postupně se prochází celý graf tak, že jsou vybírány hrany s nejnižším ohodnocením a přidávají se do předem vytvořeného seznamu.
    - Po každém přidání hrany se kontroluje, zda již minimální kostra obsahuje všechny uzly původního grafu a zda minimální kostra neobsahuje kružnice (hrany několika uzlů uzavírají kruh). Pokud jsou obsaženy všechny prvky: Minimální kostra je vytvořena. Pokud ne: Pokračuje se s přidáváním.



Obr. 27: Vývojový diagram Kruskalova hladového algoritmus. Varianta s přidáváním hran.

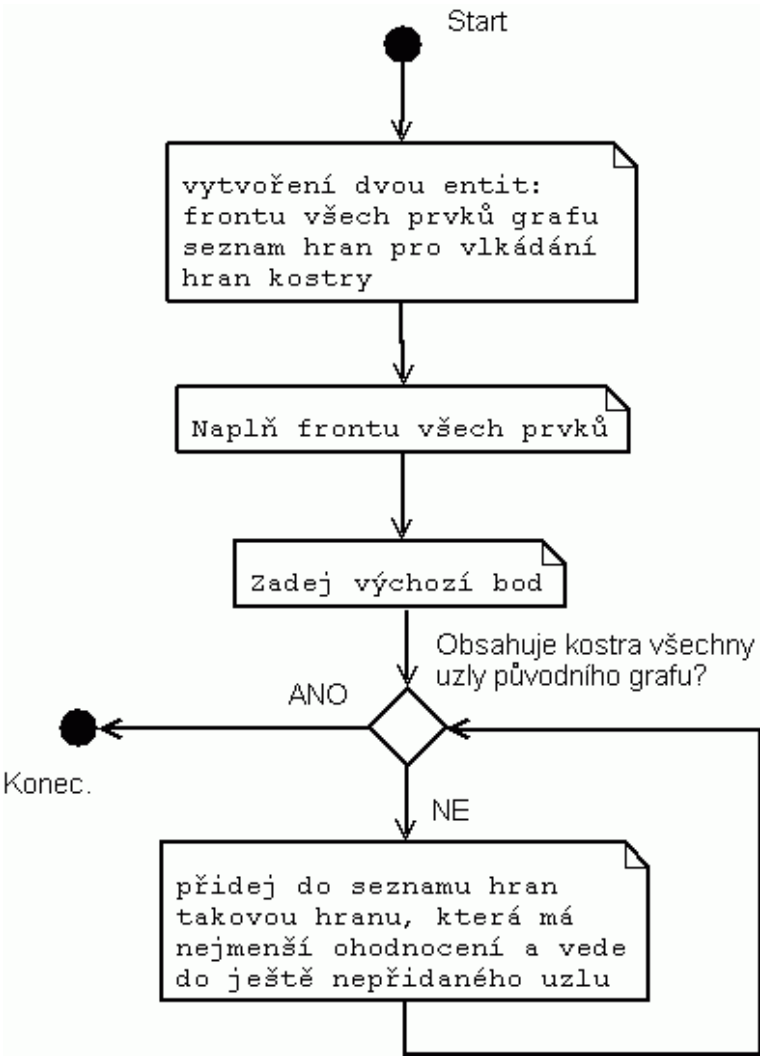
- **Postupné odebírání hran**
  - Na začátku je vytvořen seznam hran, který je naplněn všemi hranami původního grafu a celý seznam je seřazen podle velikosti ohodnocení hran.
  - Postupně se prochází celý seznam od nejvýše ohodnocených hran k nejníže ohodnoceným.
  - Při každém odebrání hrany je zkontrolováno, zda jsou v seznamu obsaženy všechny uzly původního grafu a zároveň jestli nedošlo k rozpojení grafu. Pokud jsou tato kritéria splněna: Je hrana smazána. Pokud ne: Zůstává hrana v seznamu.
  - Hrany, které v seznamu zůstanou, jsou hranami minimální kostry.



Obr. 28: Vývojový diagram Kruskalova hladového algoritmus. Varianta s odebíráním hran.

4.7.2 Jarníkův (Primův) algoritmus

Tento algoritmus pro určení minimální kostry vychází z toho, že hrany přidávané do množiny hran reprezentující podstrom jsou stále jedním podstromem. K tomuto podstromu se vždy přidává jeden nový uzel. Tento uzel, spolu s hranou, která ho připojuje k minimální kostře, má od kostry nejmenší vzdálenost. Hodnoty nejmenších vzdáleností uzlů od minimální kostry jsou postupným přidáváním nových uzlů měněny, a proto je zapotřebí minimální vzdálenosti, při každém přidání nového prvku, přepočítat. Vývojový diagram tohoto algoritmu je na obrázku 29. Oproti předchozímu algoritmu vyhledávání minimální kostry je nutné u tohoto specifikovat uzel, ve kterém je hledání spuštěno.



Obr. 29: Vývojový diagram Jarníkova (Primova) algoritmu.

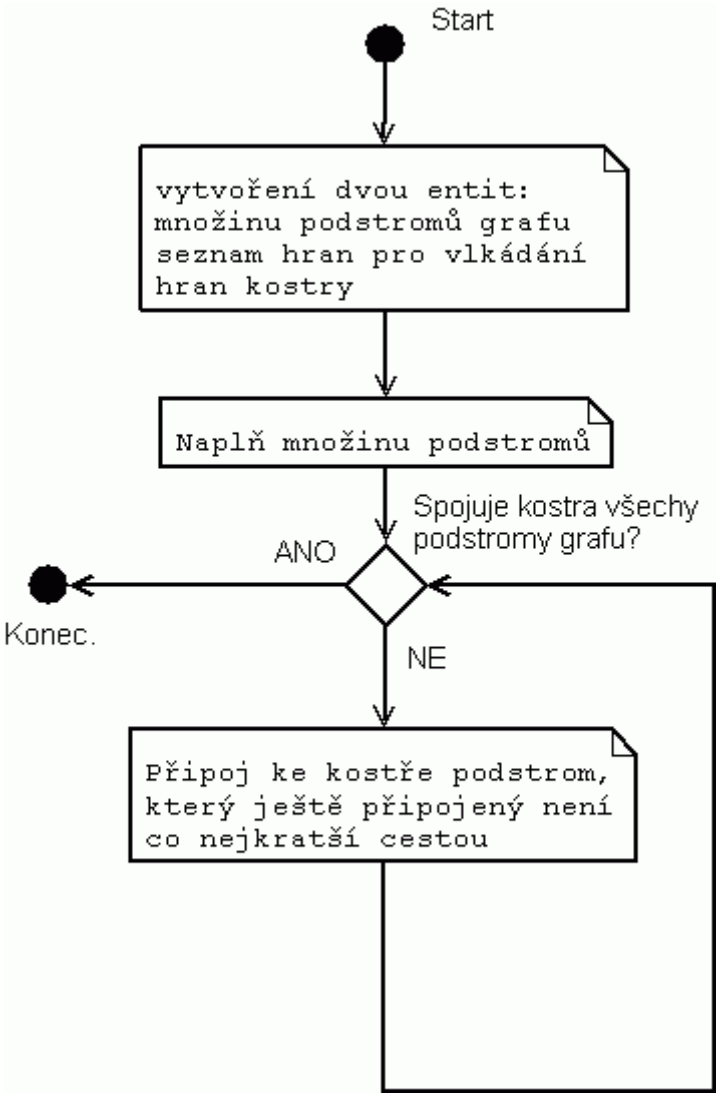
Podstata činnosti Jarníkova (Primova) algoritmu by se dala shrnut do několika následujících bodů:

- Na začátku je vytvořen seznam všech prvků grafu a seznam hran sloužící pro uložení informací podstromu, který reprezentuje minimální kostru.
- Fronta všech prvků je naplněna postupným projitím celého grafu.
- Hledání minimální kostry začíná ve výchozím bodě. A postupně se do podstromu minimální kostry přidávají hrany s nejnižší vzdáleností od dosud vytvořené kostry.
- Pokud obsahuje kostra všechny prvky jako fronta všech prvků grafu, je minimální kostra vytvořena.

4.7.3 Borůvkův algoritmus

Základní myšlenkou tohoto algoritmu je spojování podstromů minimálních koster grafu tak, že dochází ke spojování uzlů pomocí nejníže ohodnocených hran. Postupné spojování probíhá do té doby, dokud není vytvořen jeden jediný souvislý podstrom tvořící minimální kostru celého grafu. S Borůvkovým algoritmem, jehož vývojový diagram je na obrázku 30, úzce souvisí pojem jednoznačnost minimální kostry.

Dosud představené algoritmy pro vyhledávání minimální kostry pracovaly s nezáporně ohodnocenými grafy. Hrany těchto grafů mohou být ohodnoceny různě. To znamená, že více hran může být ohodnoceno stejně. Při vytváření minimálních koster z různých počátečních bodů tak neexistuje jednoznačně určená minimální kostra. Pokud je však zaručeno, že žádné dvě ani více hran v grafu nebudou mít stejné ohodnocení, je kostra vytvořená z jakéhokoliv počátečního bodu vždy stejná a tudíž i jednoznačně určená.



Obr. 30: Vývojový diagram Borůvkova algoritmu.

- Na začátku je vytvořen seznam všech prvků grafu a seznam hran sloužící pro uložení informací podstromu, který reprezentuje minimální kostru.
- Fronta všech prvků je naplněna postupným projitím celého grafu.
- Hledání minimální kostry začíná ve výchozím bodě. A postupně se do podstromu minimální kostry přidávají hrany s nejnižší vzdáleností od dosud vytvořené kostry. Tato vzdálenost je určena jednoznačně díky způsobu ohodnocení původního grafu.
- Pokud obsahuje kostra všechny prvky jako fronta všech prvků grafu, je minimální kostra vytvořena.

4.7.4 Porovnání metod pro výpočet minimální kostry

Hledání minimální kostry v grafech je velmi diskutovaným problémem z hlediska komprimace nebo úspory datových zdrojů v případě rozsáhlých grafových struktur. Jak již bylo uvedeno, obecně orientované grafy jsou velmi málo nasazovány a výpočet minimálních koster se tak v drtivé většině zaměřuje na práci s obecně ohodnocenými grafy.

Kruskalův algoritmus pro hledání minimální kostry v grafu je realizovatelný dvěma způsoby. Prvním z nich je vytváření kostry grafu z hran s nejnižším ohodnocením. Tato varianta je velmi neúsporná co se týče paměťových nároků, protože je zapotřebí nést informaci o původním grafu i o jeho kostře. Tento nedostatek řeší varianta, která odebírá z grafu hrany s nejvyšším ohodnocením hran až dokud nezůstane z grafu jen kostra. Druhá varianta Kruskalova algoritmu je paměťově méně náročná a výpočetně rychleji realizovatelná. Obě tyto metody vyžadují pro práci graf, seřazený podle ohodnocení jednotlivých hran. Řazení může u rozsáhlých grafů vést ke zvýšeným nárokům na výpočetní výkon.

Kruskalův algoritmus pracuje s grafem jako souborem hran, který je upravován. Odlišným způsobem práce s grafem je Jarníkův algoritmus. Při výpočtu minimální kostry je počítáno vždy jen se sousedy již vzniklé minimální kostry dané výchozím bodem. Při volbě různých počátečních bodů dochází k výpočtu různých koster. Vhodnou volbou počátečního bodu je tak možné docílit efektivnější komprese.

Nejjednodušším případem vyhledávání minimální kostry v grafu je situace, kdy je výpočet prováděn v jednoznačně ohodnoceném grafu. Takový graf je různě ohodnocen pro všechny hrany. To znamená, že v celém grafu neexistují dvě stejně ohodnocené hrany. Pokud bychom uvažovali tuto podmínku i u výše zmíněných algoritmů, vždy bychom dospěli ke správnému

výsledku. Při použití Borůvkova algoritmus však není možné pracovat s obecně ohodnoceným grafem.

5 Závěr

Teorie grafů má svoje uplatnění nejen na poli matematiky, ale i výpočetní techniky. Mnoho relativně odlišných úloh lze totiž řešit pomocí teorie grafů. Souborové systémy založené na binárních stromech přímo využívají znalosti vyhledávání v grafech. Vhodná volba metody vyhledávání tak může mít velmi zásadní vliv na efektivitu vyhledávání a to platí i naopak. Nevhodnou volbou vyhledávacího algoritmu může dojít k nepříjemnostem.

Celá práce je od začátku určená pro čtenáře, kteří mají s teorií grafů velmi malou nebo dokonce žádnou zkušenost. Jsou vysvětleny základní pojmy a na jednoduchých obrázcích je ukázáno, jakým způsobem je možné jednotlivé grafy zakreslit.

Druhá kapitola je věnována počítačové analýze využívané v teorii grafů. Kapitola obsahuje dvě hlavní podkapitoly. První kapitola ukazuje možnosti jakými lze uložit všechny informace o grafu. Druhá kapitola je potom věnována způsobu popisu grafů co se týče uložení grafu v paměti počítače. Každá ze zmíněných metod má svoje výhody a nevýhody. Nejlepší se ale jeví metoda, jenž používá pro uložení ukazatele. Není sice nejšetrnější ke spotřebování paměti, ale co se týče výkonu je nejefektivnější.

Poslední část práce je velmi rozlehlá a co se týče jednotlivých podkapitol, je i velmi obsáhlá. V podstatě má však za cíl jen seznámit čtenáře s teorií k jednotlivým problémům, které jsou řešeny v praktické části.

První dvě podkapitoly v poslední části se věnují problému jednosměrně a obousměrně vázaných seznamů. Ačkoli se jedná o seznamy, bývají v literatuře často označovány jako grafy. Proto je zde ukázáno jakým způsobem je možné s těmito entitami pracovat. Jedná se hlavně o metody mazání, vkládání nebo vyhledávání jednotlivých prvků seznamu. Čtenář tak získá dobrý teoretický základ pro práci jednotlivými prvky seznamu, což je velmi užitečné při definování složitějších grafů.

Další část poslední kapitoly je jakýmsi přechodem od konkrétního typu grafu k obecnému vyjádření. Podkapitola je věnována abstraktnímu datovému typu strom. Opět je čtenář seznámen s problémem práce s jednotlivými prvky grafu. Poměrně podrobně jsou rozebrány metody odstraňování položek ze stromu, jelikož se jedná v této části teorie grafů o nejsložitější problém.

Čtvrtá část poslední kapitoly už nenahlíží na graf jako na soubor jednotlivých částí, ale pouze jako na celek, aby mohly být demonstrován další problém, a to je vyhledávání prvků

v grafech. Vyhledávaní v grafech je důležité i v jiných oborech než v teorii grafů jak je uvedeno výše. Podkapitola uvádí dva základní způsoby průchodu grafem. Prohledávání do šířky a prohledávání do hloubky. Tyto dva způsoby se liší zejména ve způsobu, jaké prvky grafu jsou pro hledání zvoleny dříve. Jedná se však pouze o neinformované metody průchodu, které se ke všem částem grafu chovají naprosto stejně. Dalšími informacemi v této podkapitole jsou i další metody. Ty však z dvou hlavních vycházejí nebo je pouze rozšiřují a zefektivňují.

Předposlední část poslední kapitoly se věnuje problému hledání nejkratší cesty v grafu. Toto téma je v praxi hojně využíváno na poli informačních sítí. Čtenář je seznámen se základním problémem hledání nejkratší cesty a je mu představeno několik možností řešení. Prvním zástupcem hledání nejkratší cesty je Dijkstrův algoritmus. Tento postup výpočtu se používá v informačních sítích u *link-state* protokolů, které sledují stav komunikačních linek, a na základě těchto informací se, s využitím právě Dijkstrova algoritmu, vypočítává směrovací tabulka, která určí nejefektivnější způsob komunikace. Dalšími zástupci výpočtu nejkratší cesty jsou Floyd-Warshallův a Bellman-Fordův algoritmus, které jsou spíše teoretickou záležitostí, jelikož jejich vlastnosti neumožňují takové využití jako má Dijkstrův algoritmus.

Poslední část závěrečné kapitoly je určena k seznámení s kostrami grafů. Kostra grafu je jakýmsi zjednodušením původního grafu, jež však zachovává jeho základní vlastnosti. Tento pojem je opět využíván v informačních sítích. Existuje mnoho různých postupů upravující graf na kostru a několik základních z nich je představeno i zde. Výčet je zahájen Kruskalovým hladovým algoritmem, vytvářejícím kostru grafu pomocí odebírání nebo naopak přidávání jednotlivých hran grafu. Obě tyto metody jsou velmi efektivní a často využívané. Dalším z rodiny algoritmů vyhledávající kostru grafu je méně často využívaný, ale také velmi dobrý Jarníkův (Primův) algoritmus, který proti prvnímu specifikuje výchozí bod při určování kostry. Na závěr je představen spíše teoreticky aplikovaný Borůvkův algoritmus, jehož vlastnosti neumožňují jeho efektivní nasazování.

Literatura

[1] DEMEL, Jiří. *Grafy a jejich aplikace*. Praha : Academica, 2002. 257 s. ISBN 80-200-0990-6.

[2] DEITEL, Harvey, DEITEL, Paul. *C++ How to Program (5th Edition)* (How to Program). 5th enl. edition. New Jersey : Prentice Hall, 2005. 1536 s. ISBN 0131857576.

[3] ŠEDA, Miloš. *Teorie grafů*. Vysoké učení technické v Brně, Fakulta strojního inženýrství. 2003, Dostupné z www:

http://www.uai.fme.vutbr.cz/~mseda/TG03_MS.pdf

[4] BURGET, Radim. *Teorie grafů*, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Dostupně z www:

http://adela.utko.feec.vutbr.cz/mtin/pr/06_grafy.pdf

[5] ČERNÝ, Jakub. *Základní grafové algoritmy*, Karlova Univerzita, Katedra aplikované matematiky. Dostupné z www:

http://kam.mff.cuni.cz/~kuba/ka/

[6] KASAN, Ondřej. *Hledaní minimální kostry neorientovaného grafu*, České učení technické v Praze, Elektrotechnická fakulta. Dostupné z www:

http://www.kasan.net/fel/pjw/

[7] KONOPÍK, Miloslav, *Abstraktní datový typ graf*, Západočeská univerzita v Plzni, Katedra informatiky a výpočetní techniky. Dostupné z www:

http://www.kiv.zcu.cz/~konopik/sem/cech/index.html

[8] D. Cherition and R. E. Tarjan: *Finding minimum spanning trees*. In: SIAM Journal of Computing, 5 (Dec. 1976), pp. 724–741 [online]

[9] R. C. Prim: *Shortest connection networks and some generalisations*. In: Bell System Technical Journal, 36 (1957), pp. 1389–1401 [online]

[10] GRAHAM, Ronald Lewis; HELL, Pavol. *On the History of the Minimum Spanning Tree Problem. Annals of the History of Computing*, IEEE, 12. leden-březen 1985, roč. 7, čís. 1, s. 43-57.

[11] CANNY, John, A Computational Approach to Edge-Detection In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679-700, 1986.

PROHLÁŠENÍ

Prohlašuji, že svůj semestrální projekt na téma Teorie grafů – implementace vybraných problémů jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

podpis autora

Přílohy

Příloha 1: Obsah CD

Příloha 2: CD s vytvořenou aplikací a zdrojovými kódy

Příloha 1: Obsah CD

Na přiloženém CD je umístěna webová aplikace spolu se zdrojovými kódy vytvořenými v rámci diplomové práce. Přiloženy jsou i soubory vytvořené v rámci předmětu semestrální projekt.

- Semestrální projekt

  Tato složka obsahuje jednoduché aplikace vytvořené v jazyce C++ pro potřeby předmětu semestrální projekt. Mezi tyto aplikace patří: realizace jednosměrně vázaného seznamu, obousměrně vázaného seznamu a binární vyhledávací strom.

- Webová aplikace

  V tomto adresáři se nachází výuková aplikace demonstrující některé základní problémy teorie grafů. Součástí webové aplikace je i několik appletů vytvořených v jazyce JAVA™ .

- Zdrojové kódy

  V posledním adresáři se nachází všechny zdrojové kódy všech vytvořených aplikací.